Getting Started with Graphics for iPhone OS

Session 302

These are confidential sessions—please refrain from streaming, blogging, or taking pictures.

Getting Started with Graphics for iPhone OS

Great Graphics Fast

Assana Fard
Quartz Technologies

Agenda

- Different tools for different applications
- Graphics technologies on the iPhone
- How to choose what's right for you
- Q&A

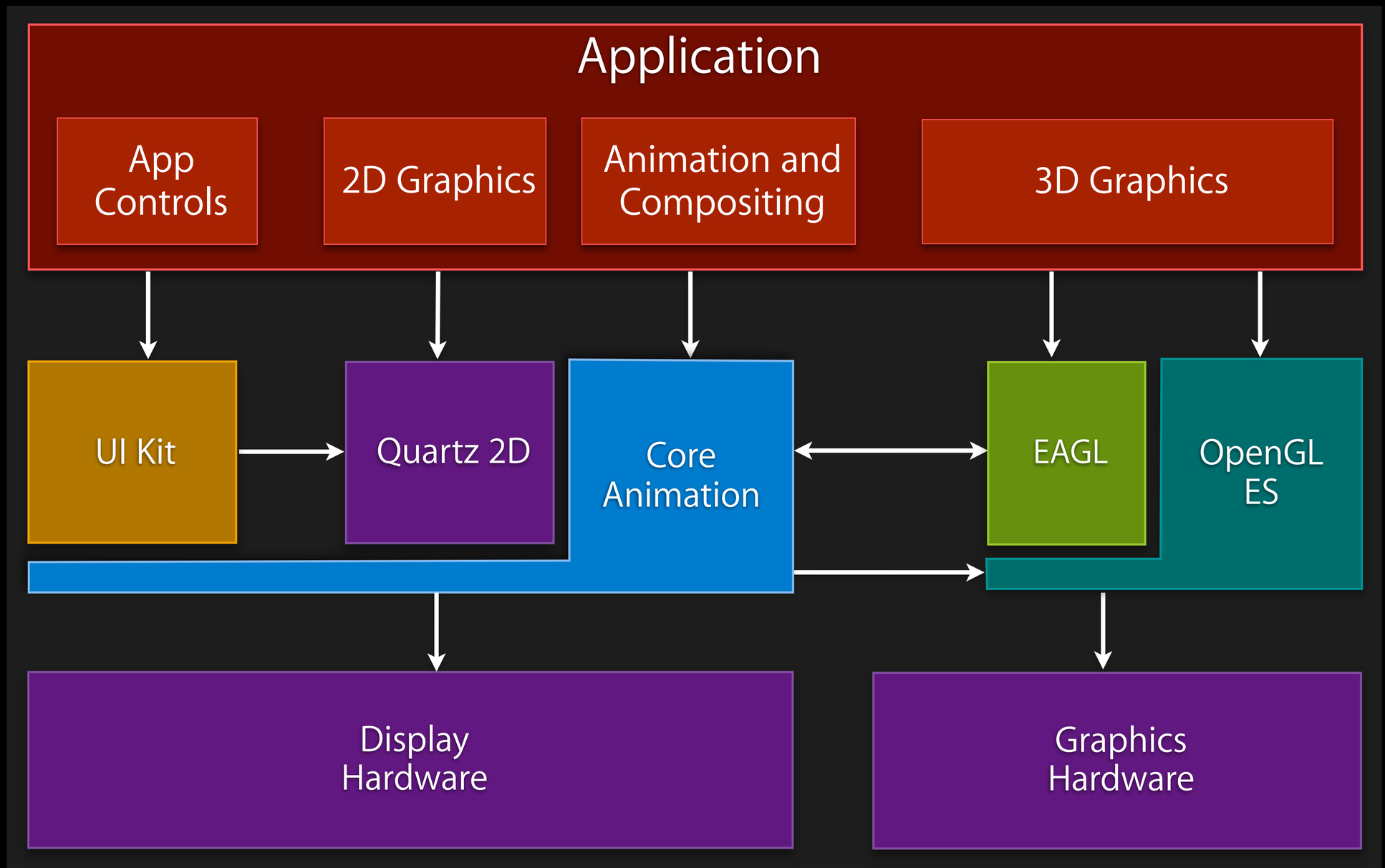
UIKit Graphics

Quartz 2D

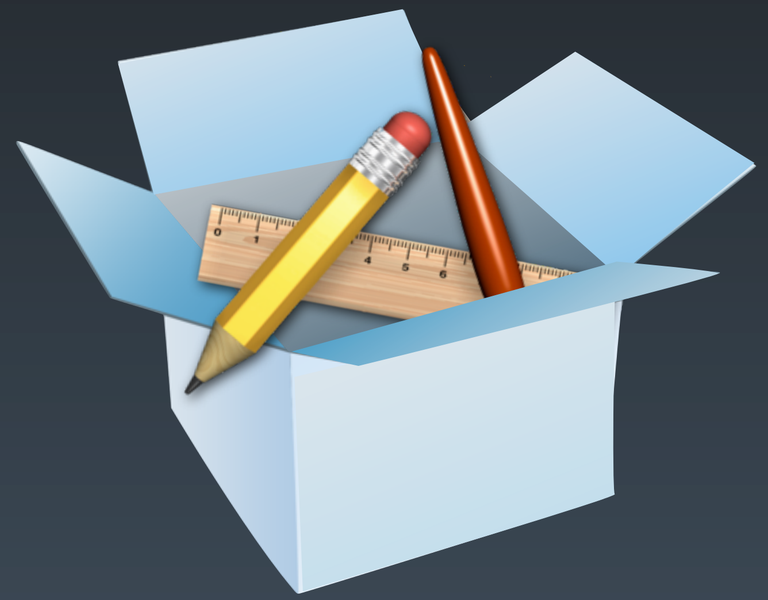
Core Animation

OpenGL ES

Graphics Architecture



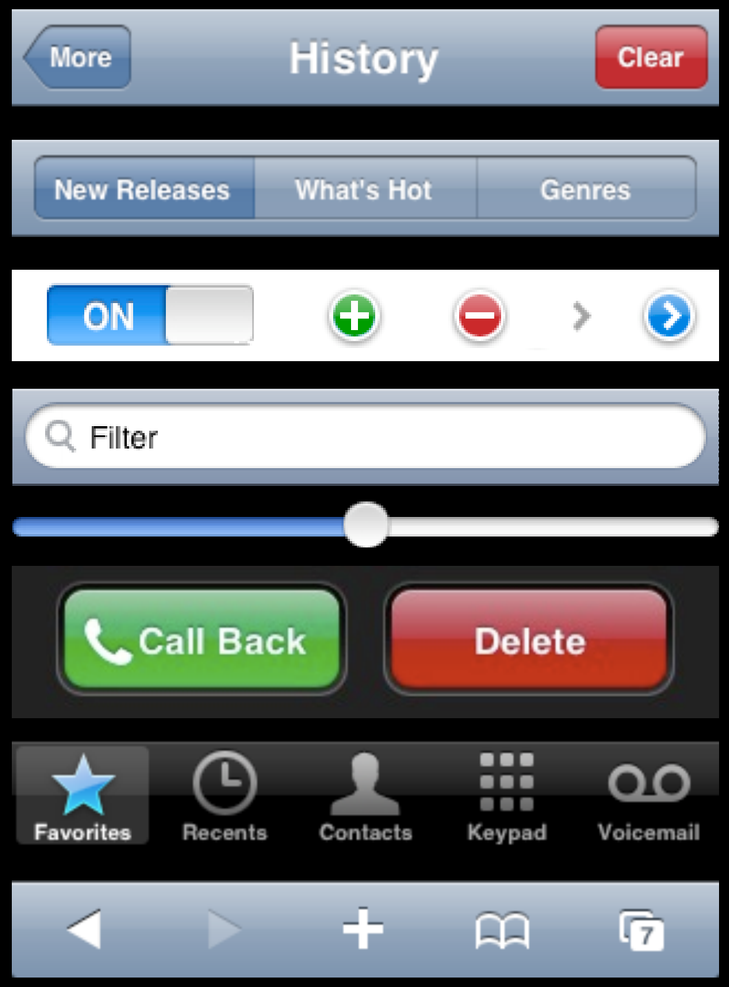
UIKit



UIKit

The heart of Cocoa Touch

- User interface
- Events
- Views



UIKit

User Interface

- Views
  - Standard UI
  - Consistent behavior
  - Layer-backed
- Available in Interface Builder
- Available from code

Graphics in UIKit

- Highest-level interface
  - For typical graphics operations in your UI
    - Drawing images
    - Setting colors
    - Filling rectangles
  - For animation of your UI
    - Move, scale, rotate
    - Fade in, fade out
    - Transitions
  - Custom drawing

UIKit Graphics

Focused set of classes for graphics

- UIImage—class for image data
- UIColor—basic support for device colors
- UIFont, UILabel—classes for drawing text
- UIScreen—basic information about the screen

Graphics utility functions

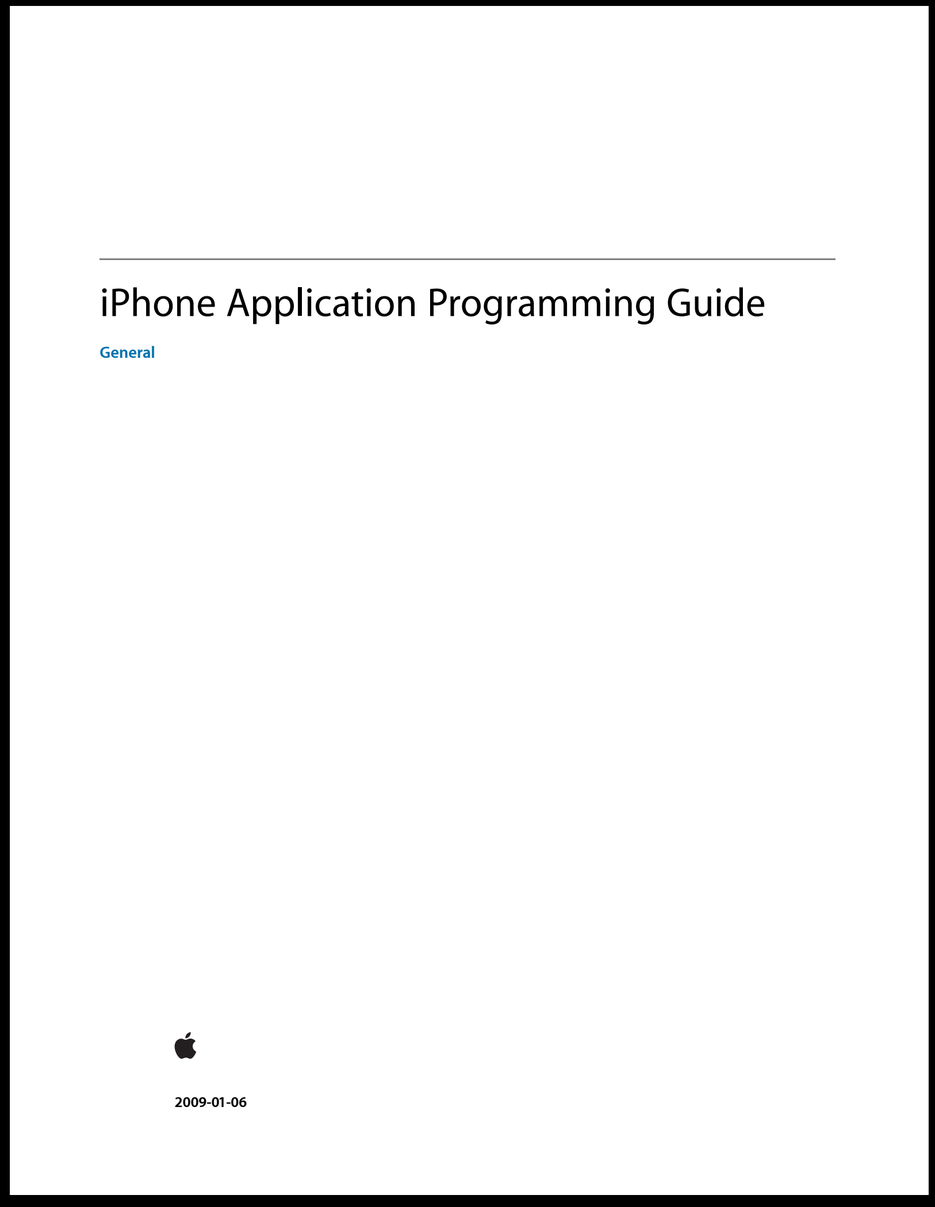
- Draw filled rectangles, outlines, and clipping
- Generate a PNG or JPEG from a UIImage
- Retrieve screen info, graphics context

Demo

UIKit

Learning More

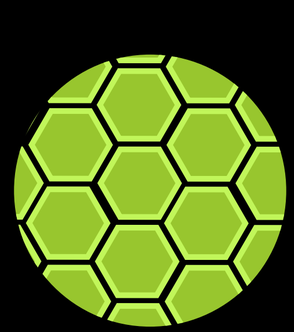
- iPhone Application Programming Guide
- iPhone Human Interface Guidelines
- iPhone OS Technology Overview

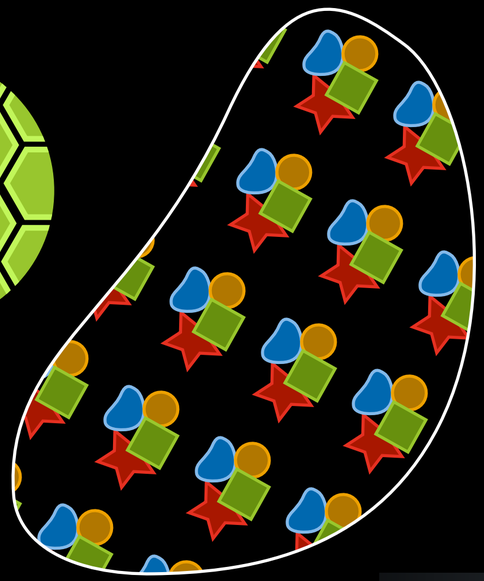


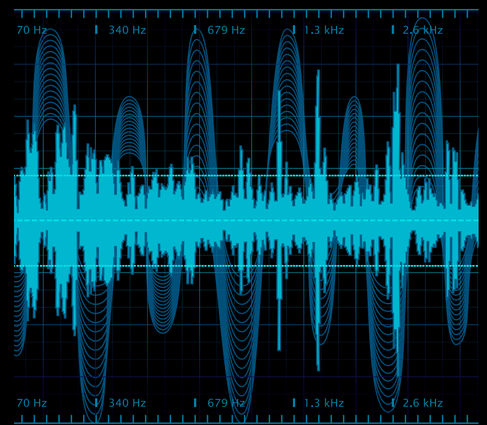
Quartz 2D

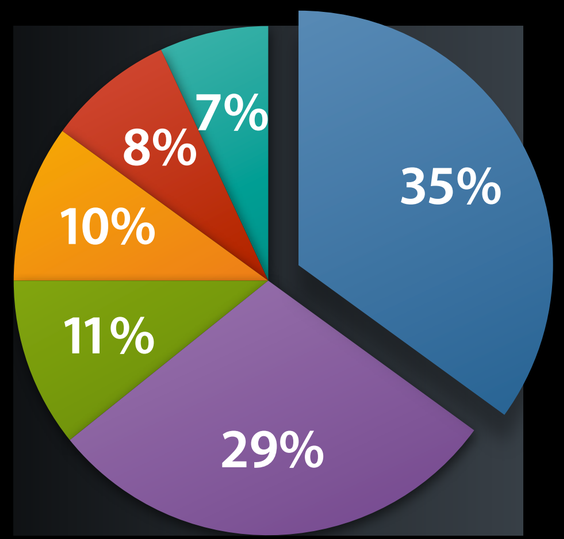
Quartz 2D

The heart of UIKit graphics









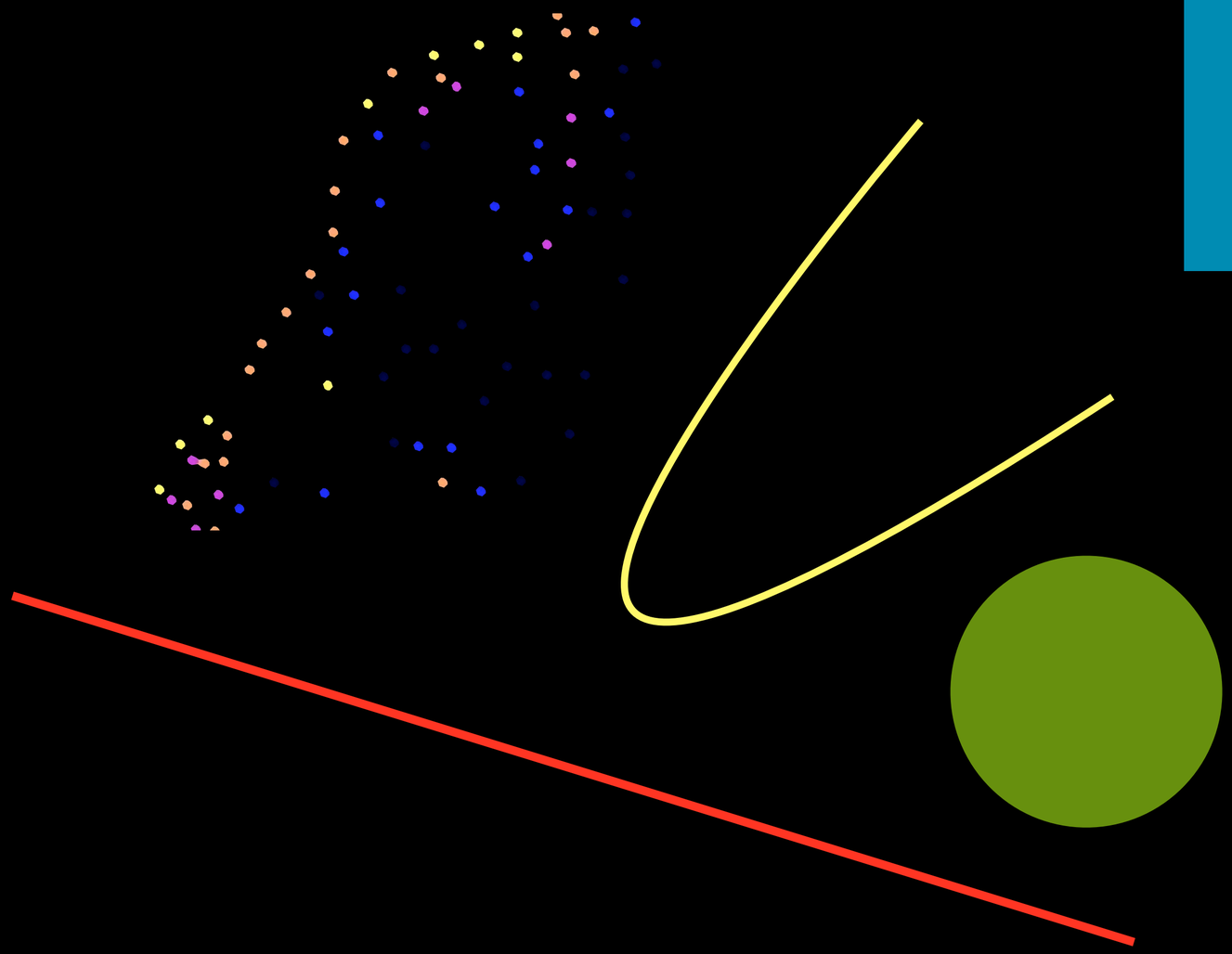
Quartz 2D

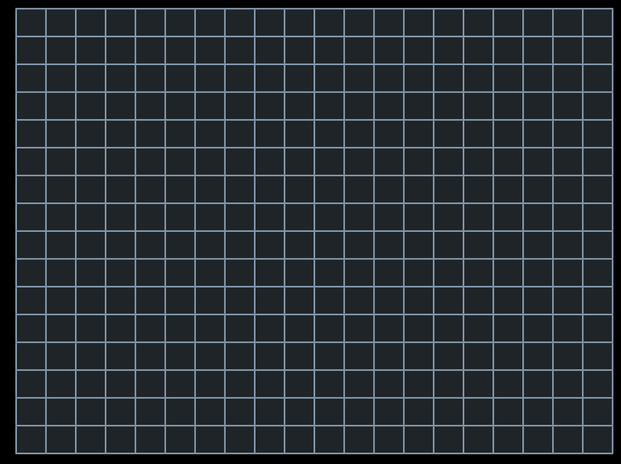
The heart of UIKit graphics

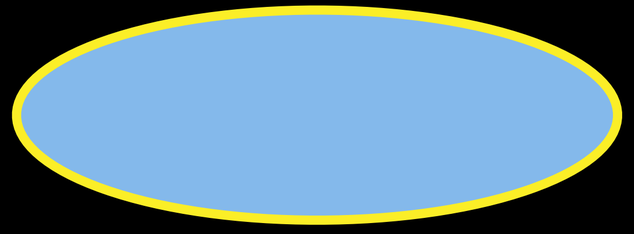
- Sophisticated 2D drawing engine
  - Vector Geometry: Path-based lines and strokes
  - Filled shapes, patterns, gradients
  - Text, Images, PDFs
- Used by higher-level application services
  - UI Kit
- Also known as “CoreGraphics”

Basic Primitives

Points, lines, arcs, curves, rectangles

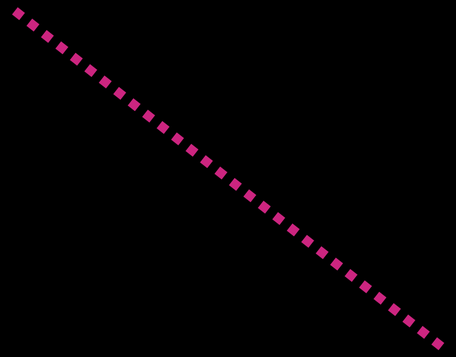


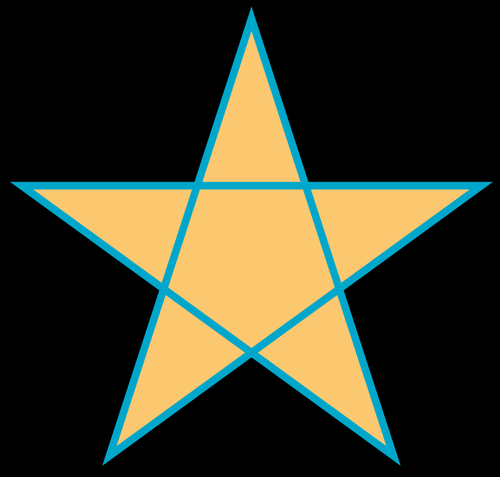




CGPathRef

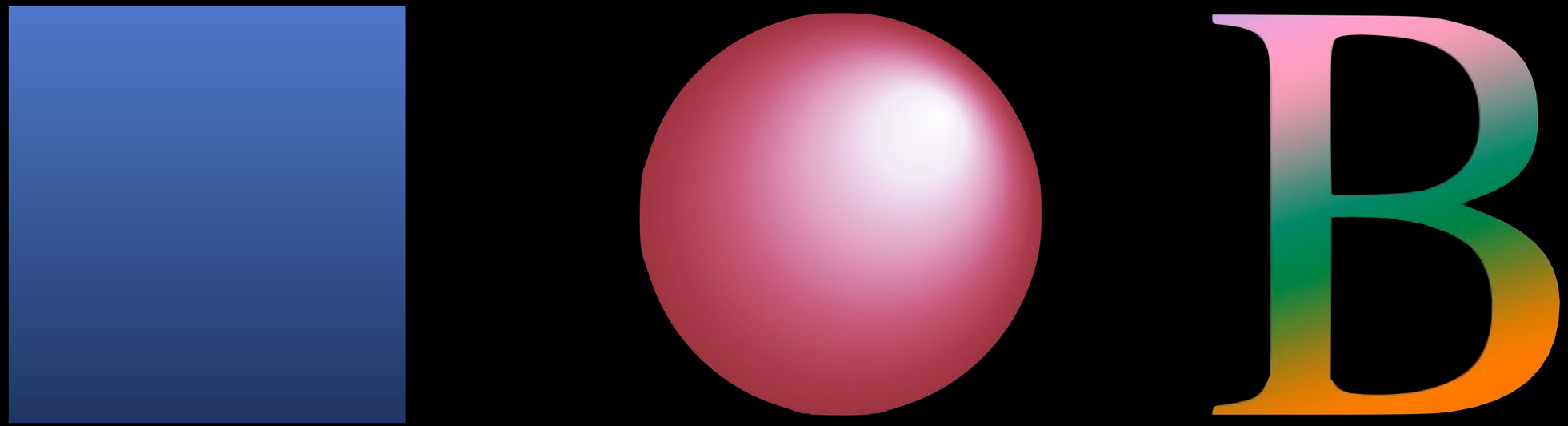
Lines and curves





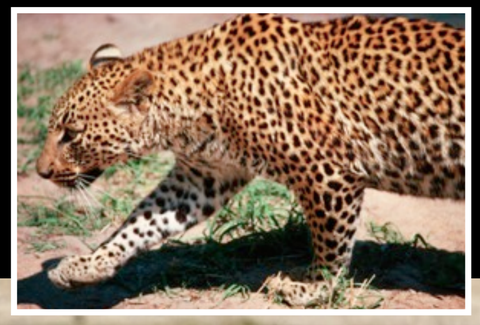
CGGradientRef

Radial and linear



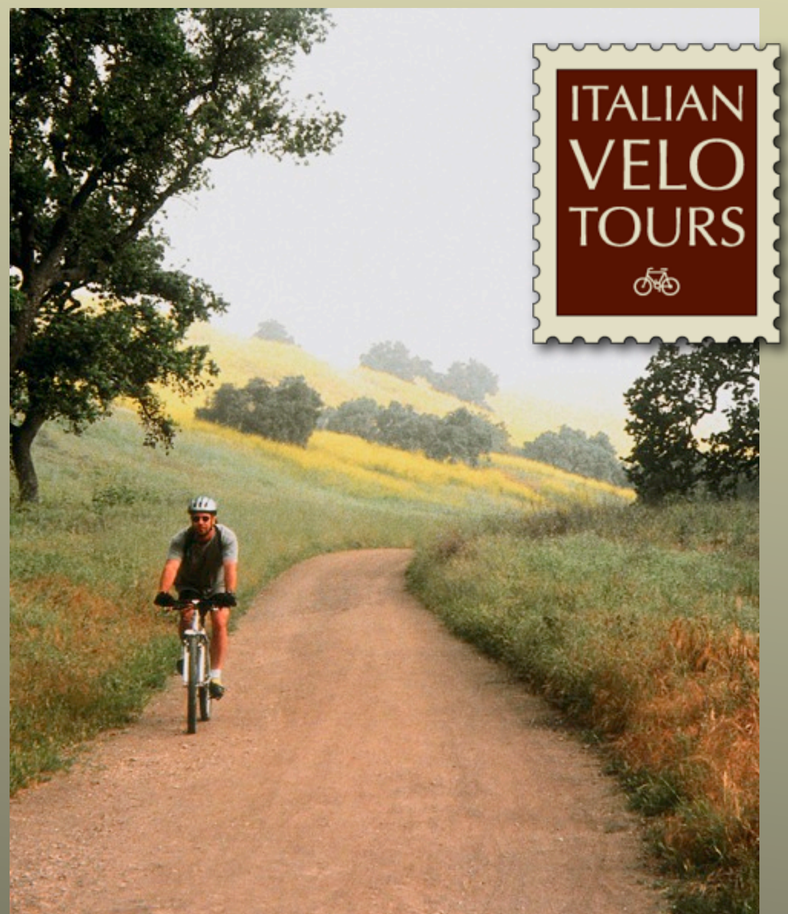
CGImageRef

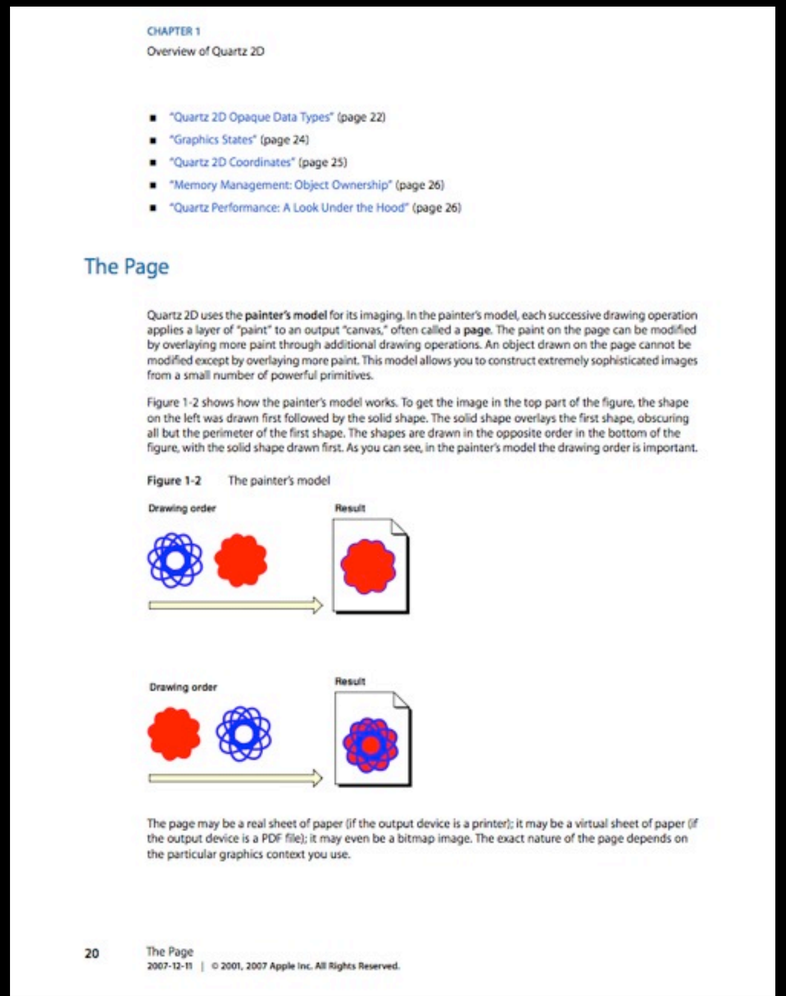
PNG, TIFF, JPEG, GIF, BMP, ICO, CUR, XBM



CGPDFDocumentRef

Single or multi-page PDF documents





CHAPTER 1
Overview of Quartz 2D

- "Quartz 2D Opaque Data Types" (page 22)
- "Graphics States" (page 24)
- "Quartz 2D Coordinates" (page 25)
- "Memory Management: Object Ownership" (page 26)
- "Quartz Performance: A Look Under the Hood" (page 26)

The Page

Quartz 2D uses the **painter's model** for its imaging. In the painter's model, each successive drawing operation applies a layer of "paint" to an output "canvas," often called a **page**. The paint on the page can be modified by overlaying more paint through additional drawing operations. An object drawn on the page cannot be modified except by overlaying more paint. This model allows you to construct extremely sophisticated images from a small number of powerful primitives.

Figure 1-2 shows how the painter's model works. To get the image in the top part of the figure, the shape on the left was drawn first followed by the solid shape. The solid shape overlays the first shape, obscuring all but the perimeter of the first shape. The shapes are drawn in the opposite order in the bottom of the figure, with the solid shape drawn first. As you can see, in the painter's model the drawing order is important.

Figure 1-2 The painter's model

The page may be a real sheet of paper (if the output device is a printer); it may be a virtual sheet of paper (if the output device is a PDF file); it may even be a bitmap image. The exact nature of the page depends on the particular graphics context you use.

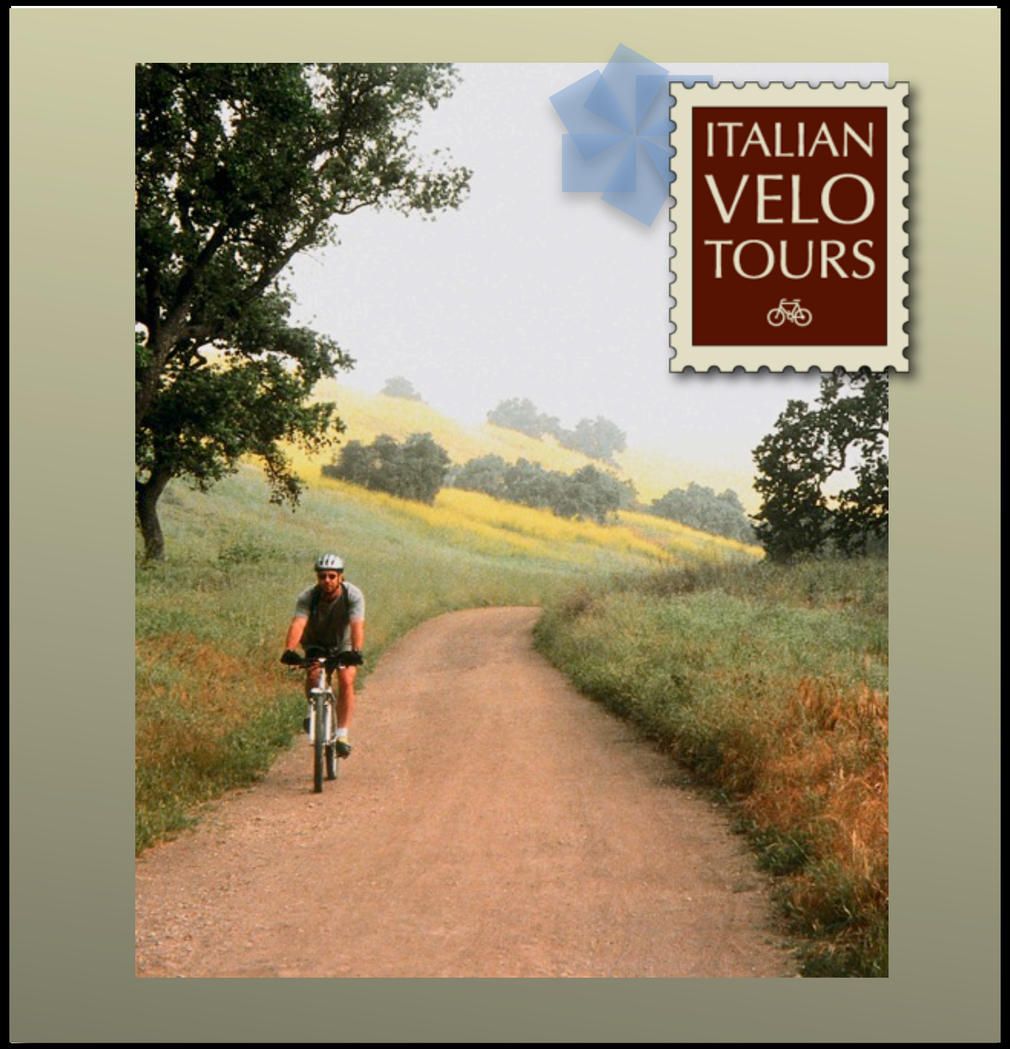
2007-12-11 | © 2001, 2007 Apple Inc. All Rights Reserved.

Painter's Model

Previous contents overdrawn

Painter's Model

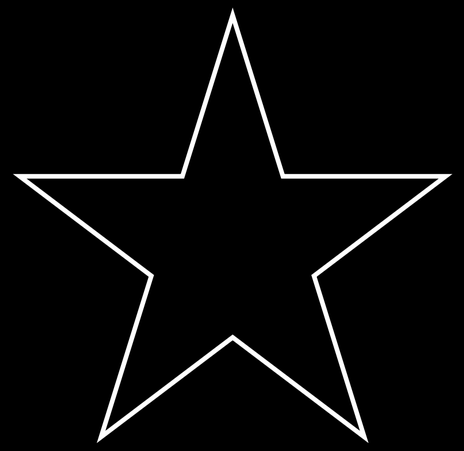
Previous contents overdrawn



CGContexts

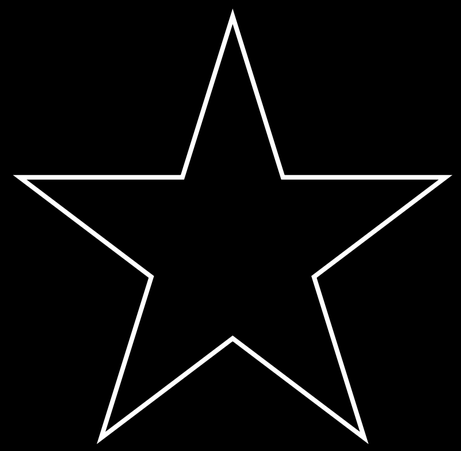
The Heart of Quartz 2D

- The destination for Quartz 2D drawing operations
- Used by other application services for drawing
  - UIKit
  - Core Animation
- A container for “graphics state”



Graphics State

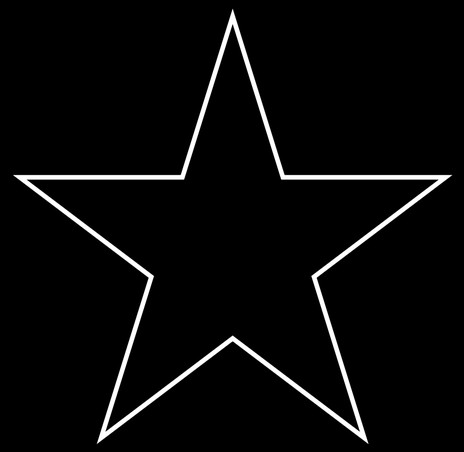
Parameters that control drawing



Graphics State

Parameters that control drawing

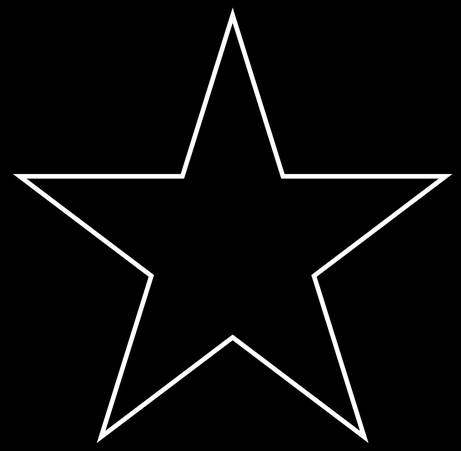
- Fill color



Graphics State

Parameters that control drawing

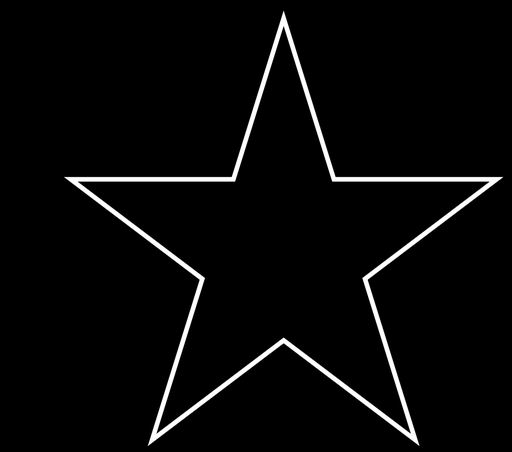
- Fill color
- Stroke color



Graphics State

Parameters that control drawing

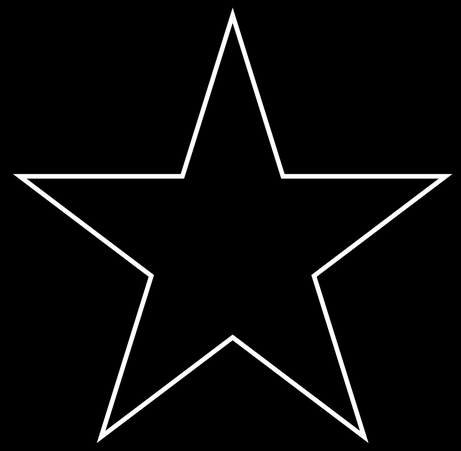
- Fill color
- Stroke color
- Coordinate space transformation (CTM)



Graphics State

Parameters that control drawing

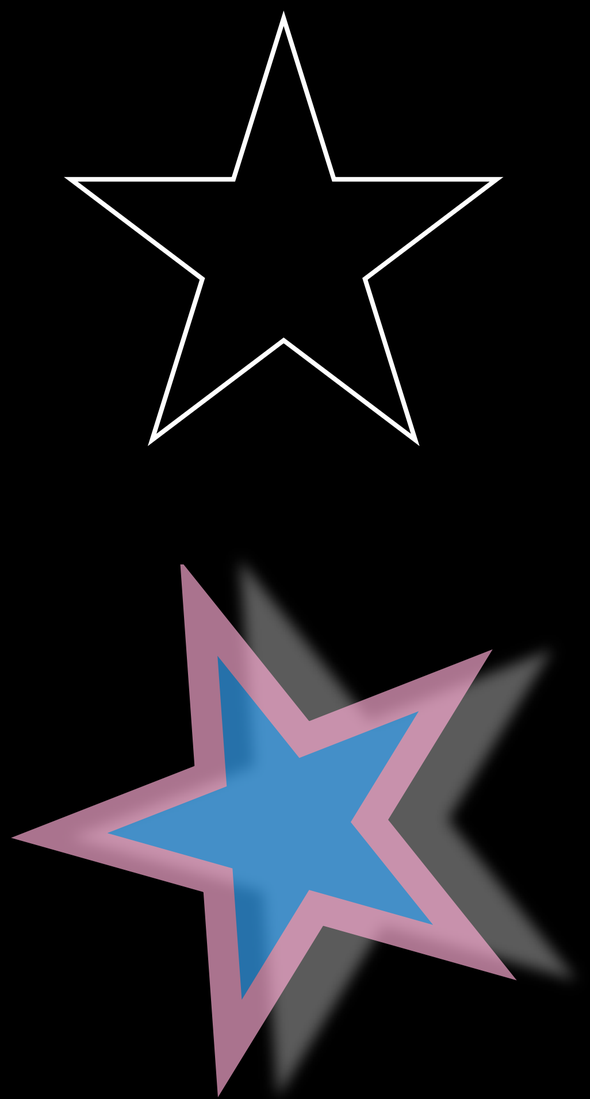
- Fill color
- Stroke color
- Coordinate space transformation (CTM)
- Line width, join, cap, dash, miter limit



Graphics State

Parameters that control drawing

- Fill color
- Stroke color
- Coordinate space transformation (CTM)
- Line width, join, cap, dash, miter limit
- Shadow



Graphics State

Parameters that control drawing

- Fill color
- Stroke color
- Coordinate space transformation (CTM)
- Line width, join, cap, dash, miter limit
- Shadow
- Global alpha
- …and more!

Getting a CGContext from UIKit

Inside UIView's drawRect: method

- Customize your drawing in a subclass of UIView
- Implement the "drawRect" method

```
-(void) drawRect:(NSRect)rect
{
    CGContextRef  myContext = UIGraphicsGetCurrentContext();

   // Your custom drawing code here
}
```

Core Graphics Drawing Example

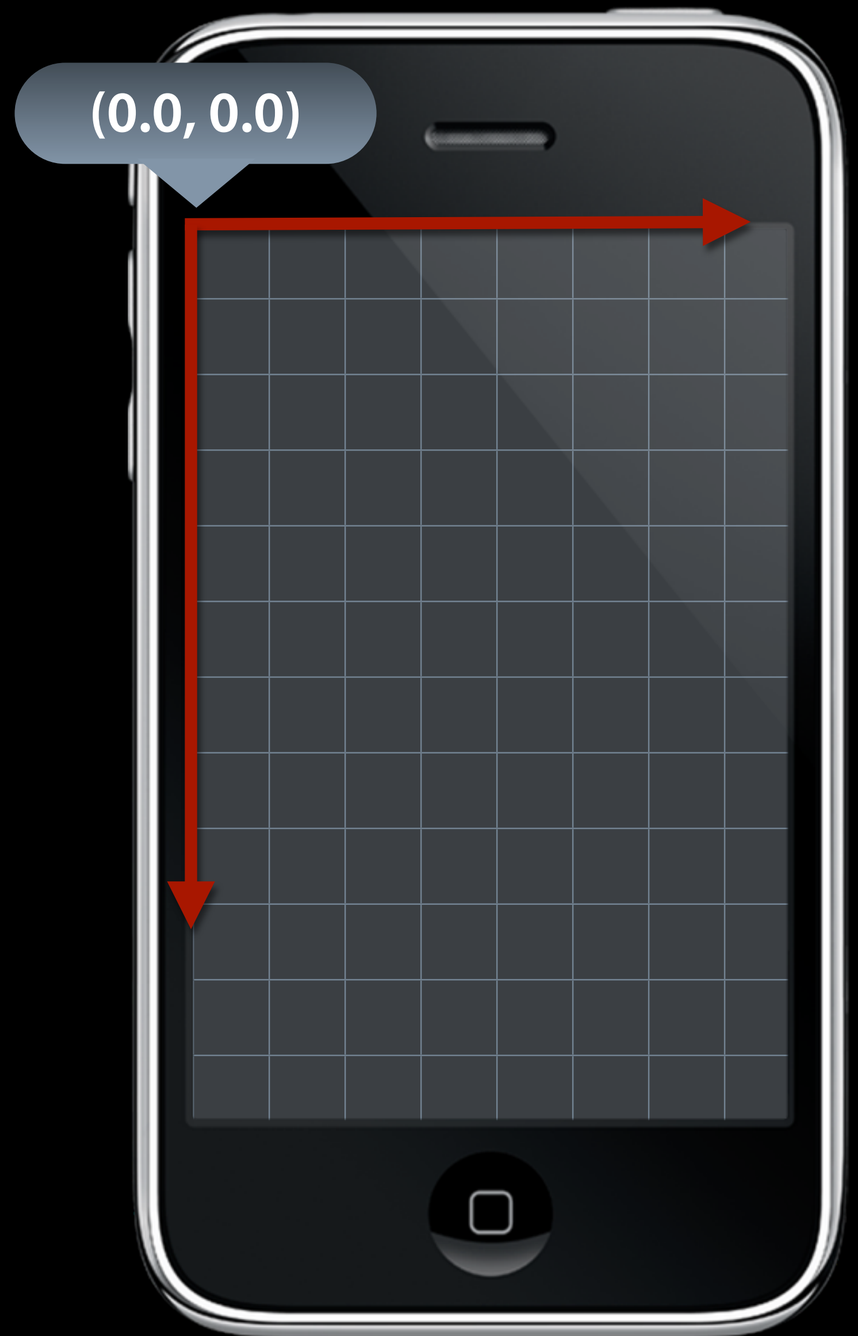
Inside UIView's drawRect: method

```
-(void) drawRect:(CGRect)rect
{
    CGContextRef  myContext = UIGraphicsGetCurrentContext();

    CGRect myRect = CGRectMake(40, 40, 240, 120);

    CGContextSetRGBFillColor(myContext, 0.0, 0.0, 1.0, 1.0);
    CGContextFillRect(myContext, myRect);

    CGContextSetRGBStrokeColor(myContext, 0.0, 1.0, 0.0, 1.0);
    CGContextStrokeRectWithWidth(myContext, myRect, 10);
}
```


(0.0, 0.0)

Core Graphics Drawing Example

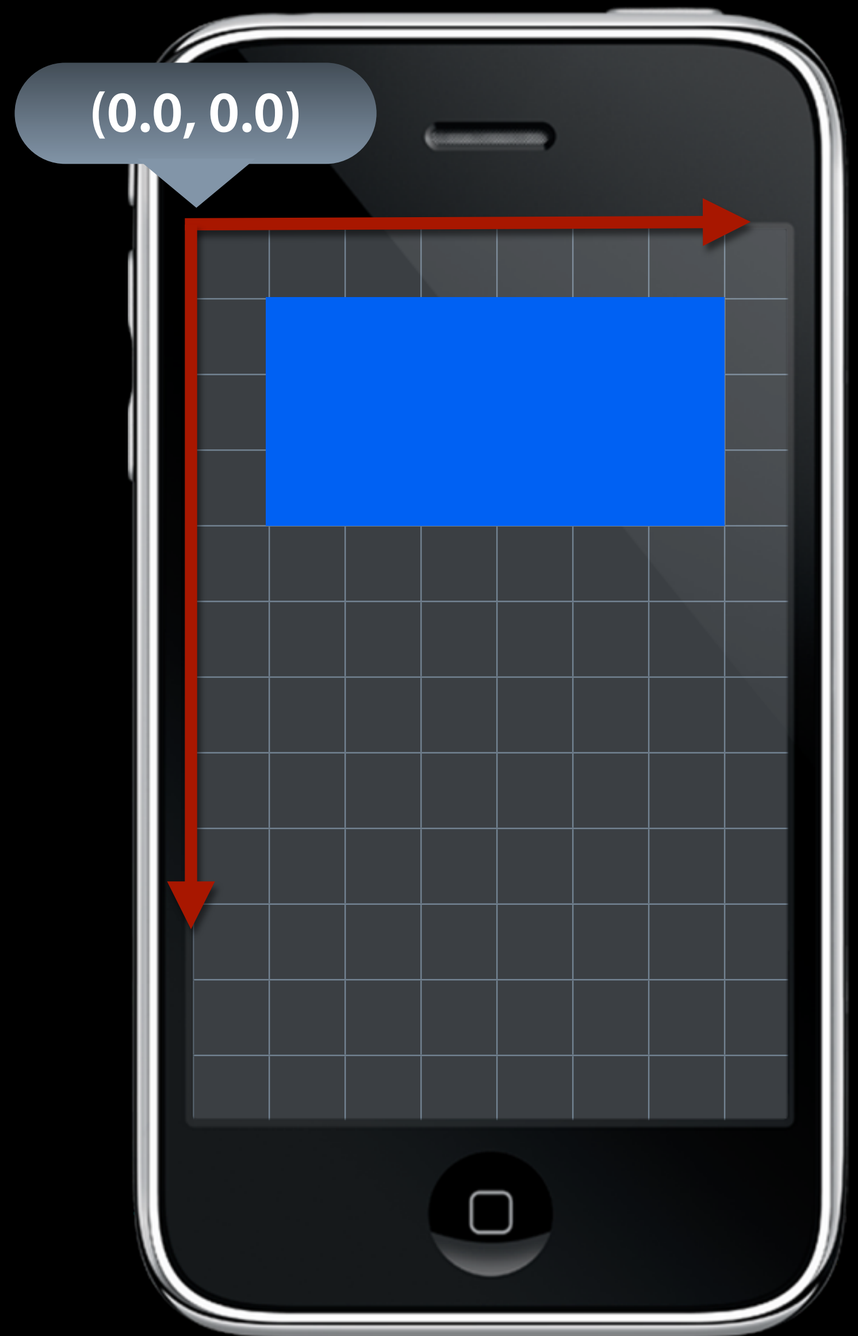
Inside UIView's drawRect: method

```
-(void) drawRect:(CGRect)rect
{
    CGContextRef  myContext = UIGraphicsGetCurrentContext();

    CGRect myRect = CGRectMake(40, 40, 240, 120);

    CGContextSetRGBFillColor(myContext, 0.0, 0.0, 1.0, 1.0);
    CGContextFillRect(myContext, myRect);

    CGContextSetRGBStrokeColor(myContext, 0.0, 1.0, 0.0, 1.0);
    CGContextStrokeRectWithWidth(myContext, myRect, 10);
}
```


(0.0, 0.0)

Core Graphics Drawing Example

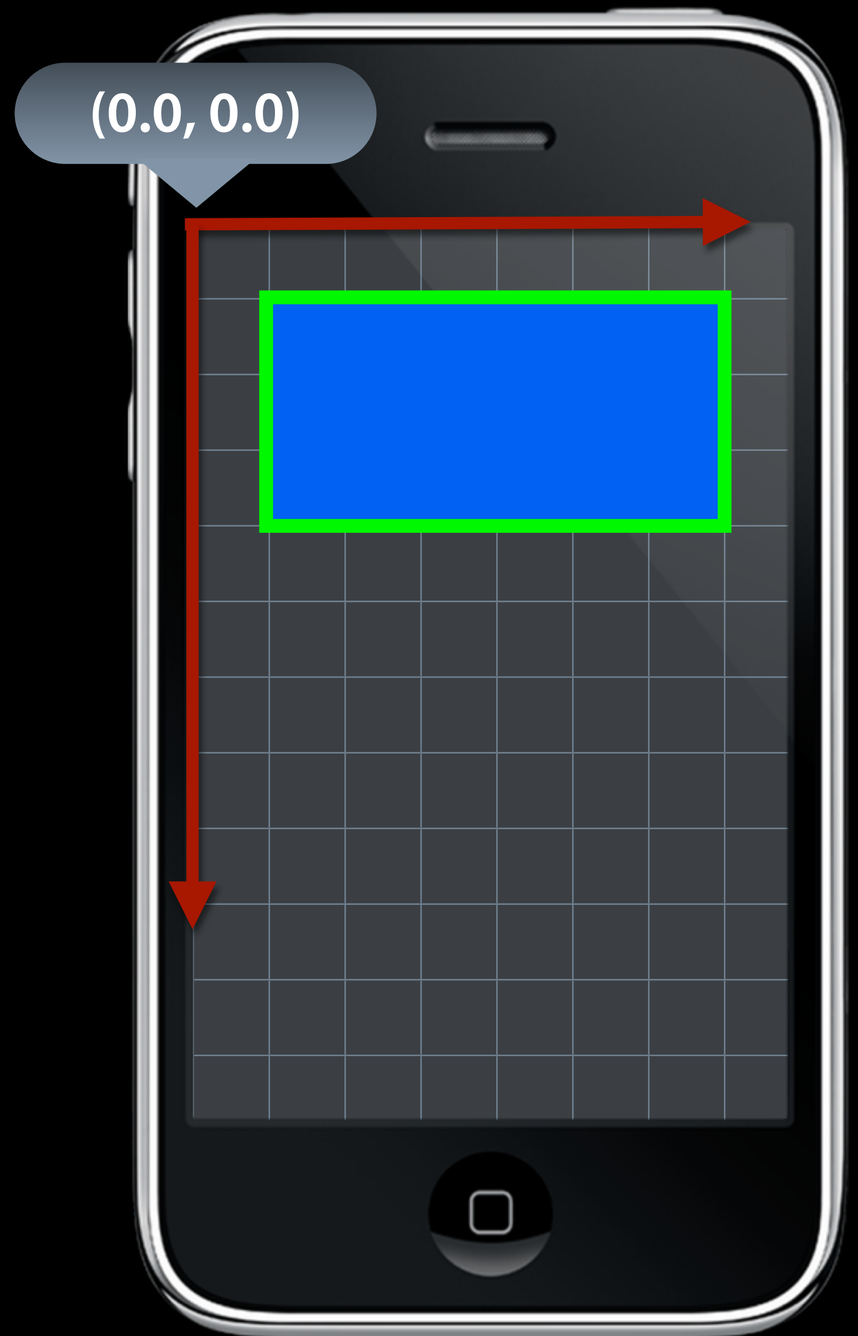
Inside UIView's drawRect: method

```
-(void) drawRect:(CGRect)rect
{
    CGContextRef  myContext = UIGraphicsGetCurrentContext();

    CGRect myRect = CGRectMake(40, 40, 240, 120);

    CGContextSetRGBFillColor(myContext, 0.0, 0.0, 1.0, 1.0);
    CGContextFillRect(myContext, myRect);

    CGContextSetRGBStrokeColor(myContext, 0.0, 1.0, 0.0, 1.0);
    CGContextStrokeRectWithWidth(myContext, myRect, 10);
}
```

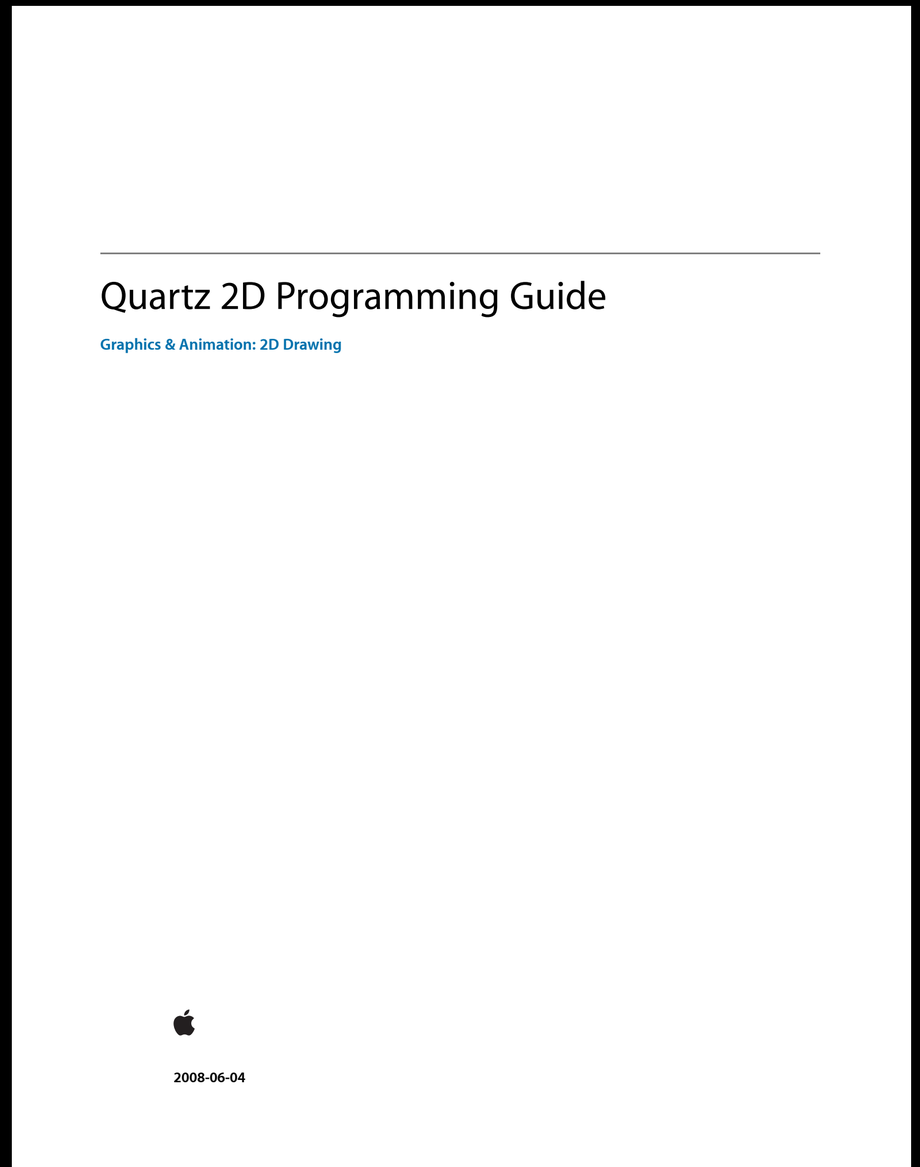

(0.0, 0.0)

Demo

Quartz 2D

Learning More

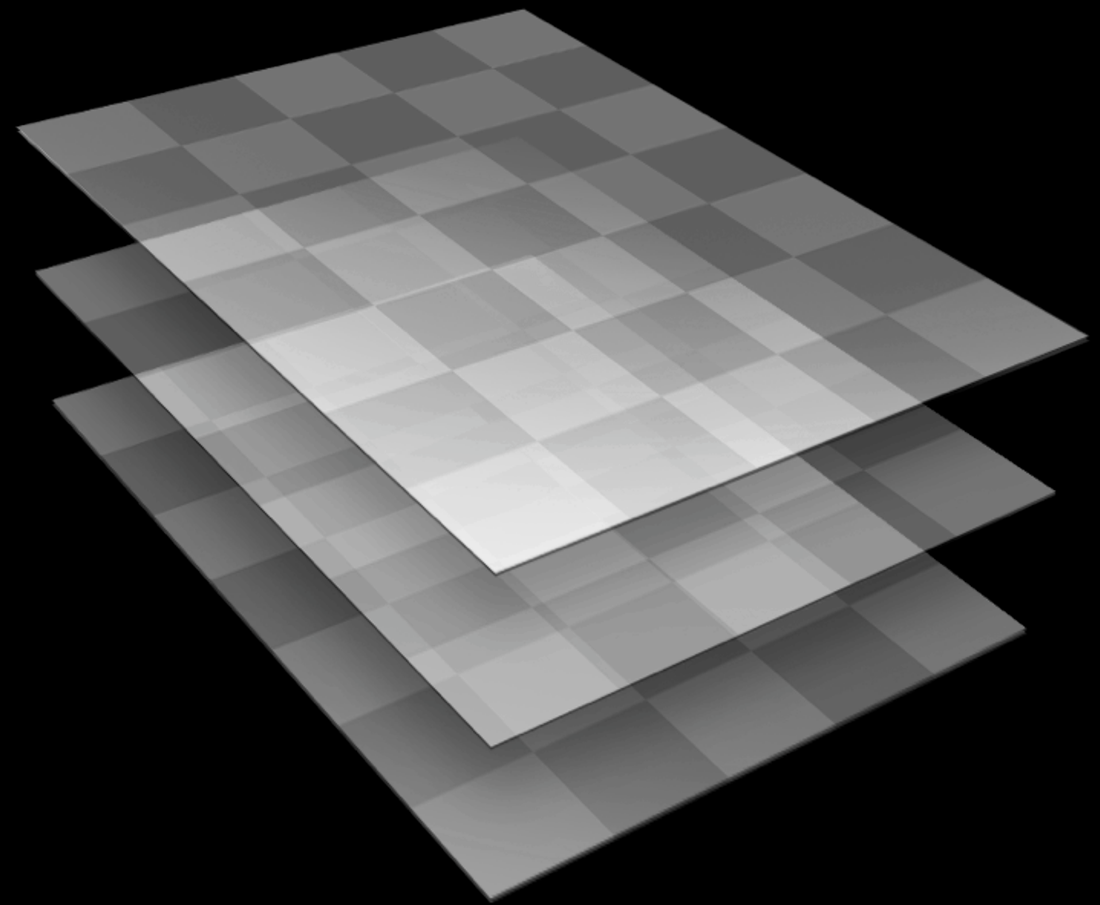
- Quartz2D Programming Guide
- “Programming with Quartz”



Core Animation

Core Animation

- Layer-based animation framework
- Easy programming model
  - Layers are the building blocks
  - Automatic transitions between states
  - Transparency effects
- Supports multiple media types
  - Text, 2D graphics, OpenGL ES, and video
- GPU accelerated and multicore enhanced
- Delivers high production value experiences



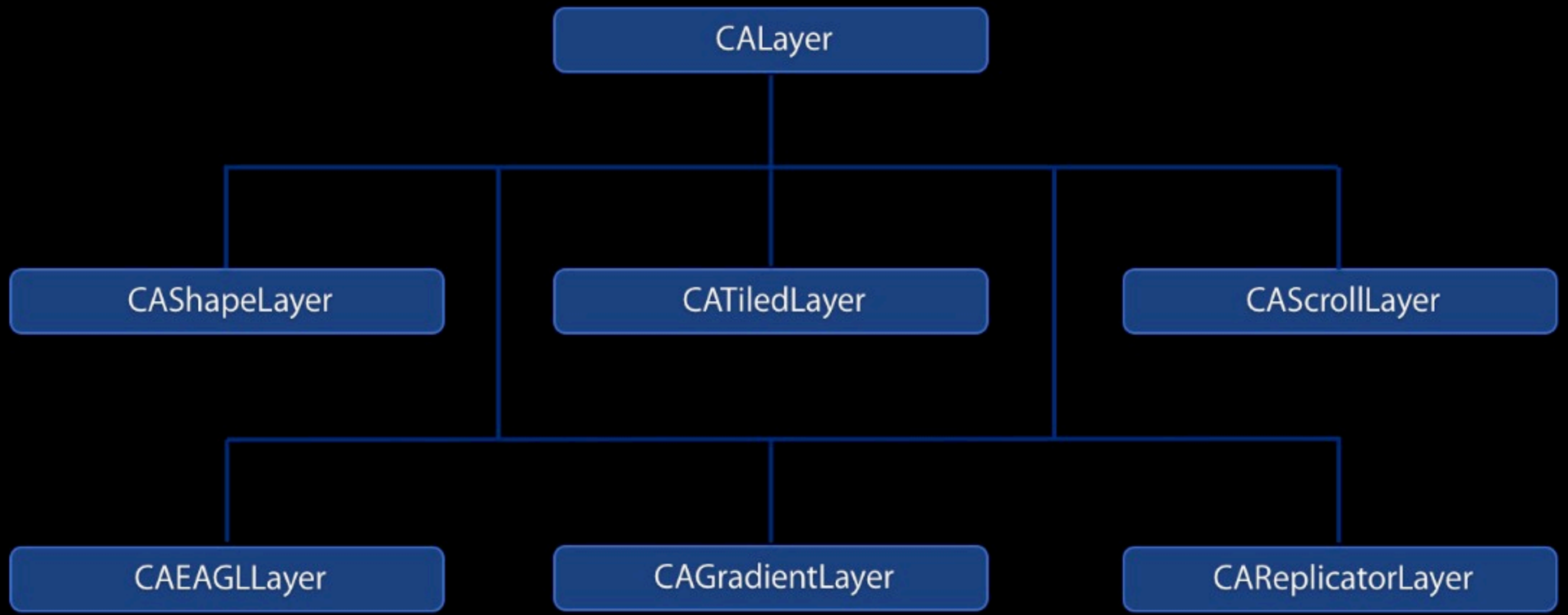
CALayer

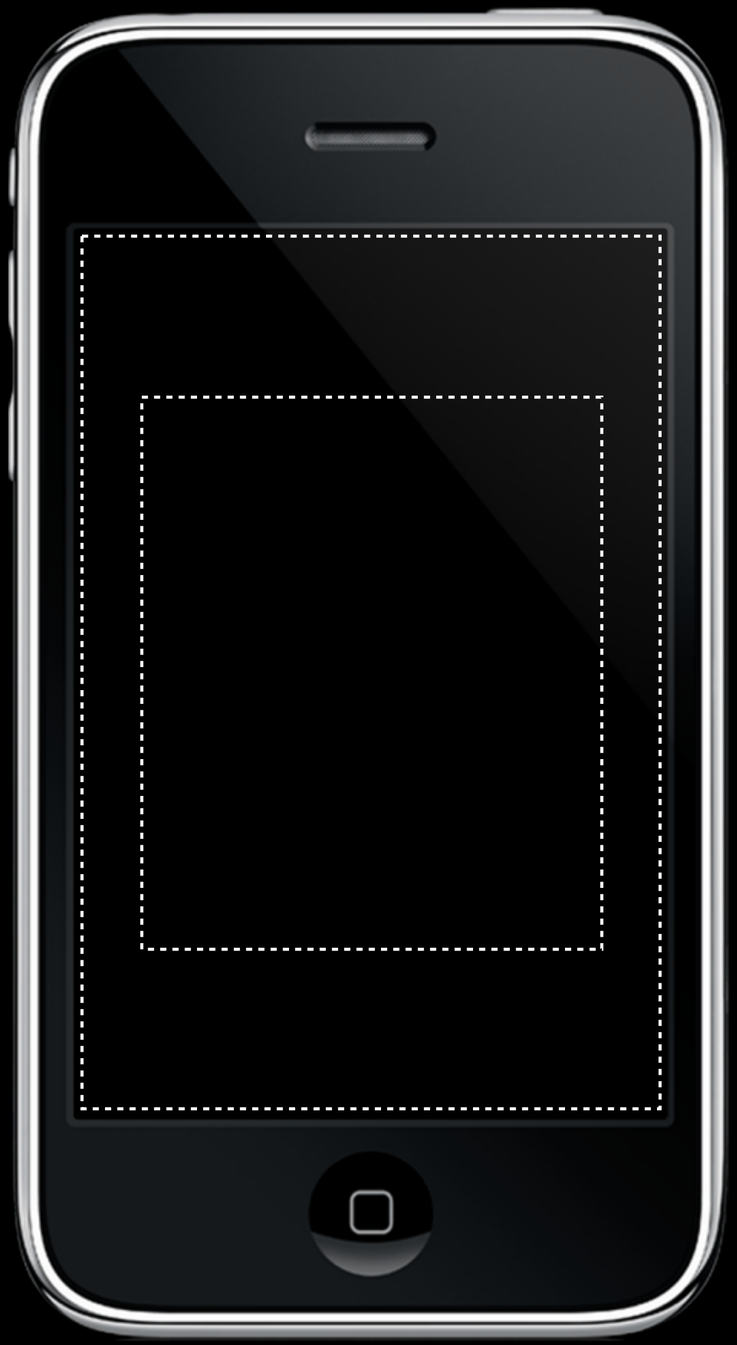
The heart of CoreAnimation

- Layers are 2D rectangles living in a 3D space
- Containers for animation
- Content comes from Quartz 2D or OpenGL ES
- Property changes invoke an implicit animation

CALayers

Core Animation's hierarchy of layer classes





Layer Properties

CALayer

- Geometry

Layer Properties

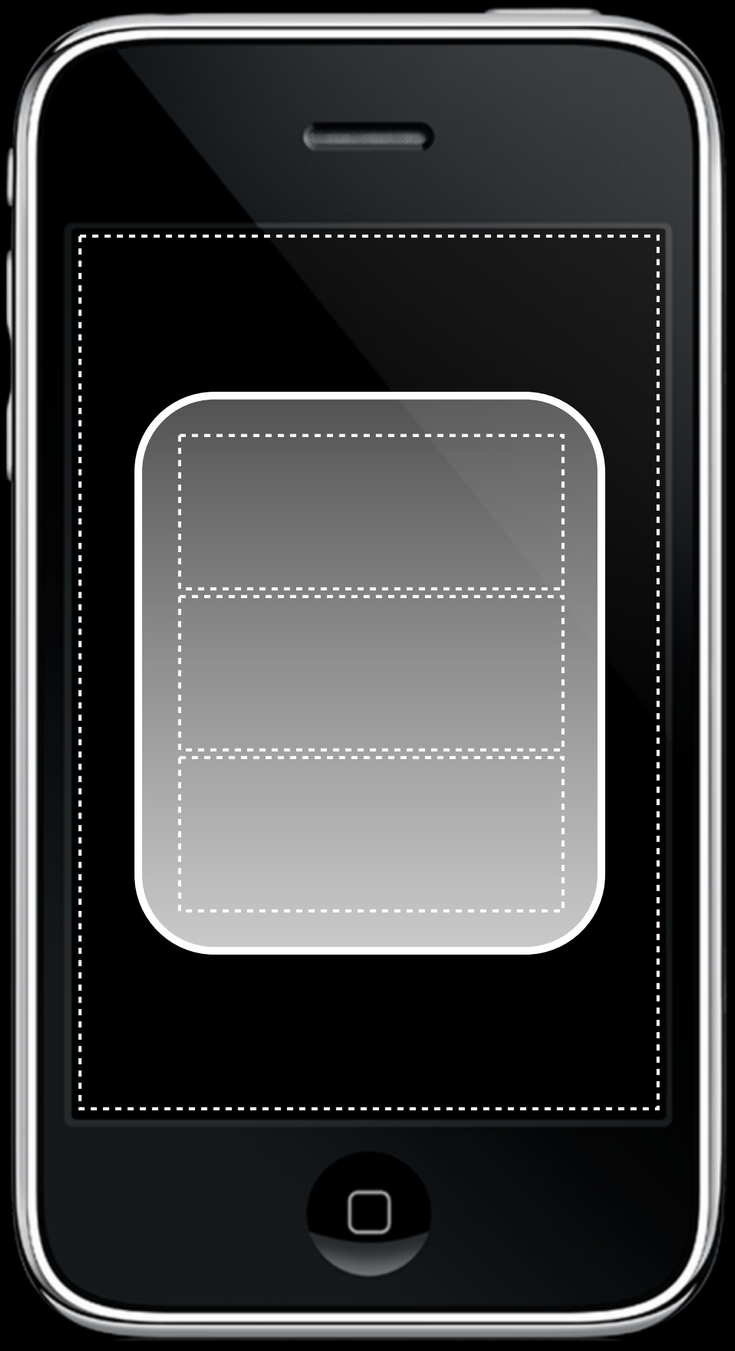
CALayer

- Geometry
- Background

Layer Properties

CALayer

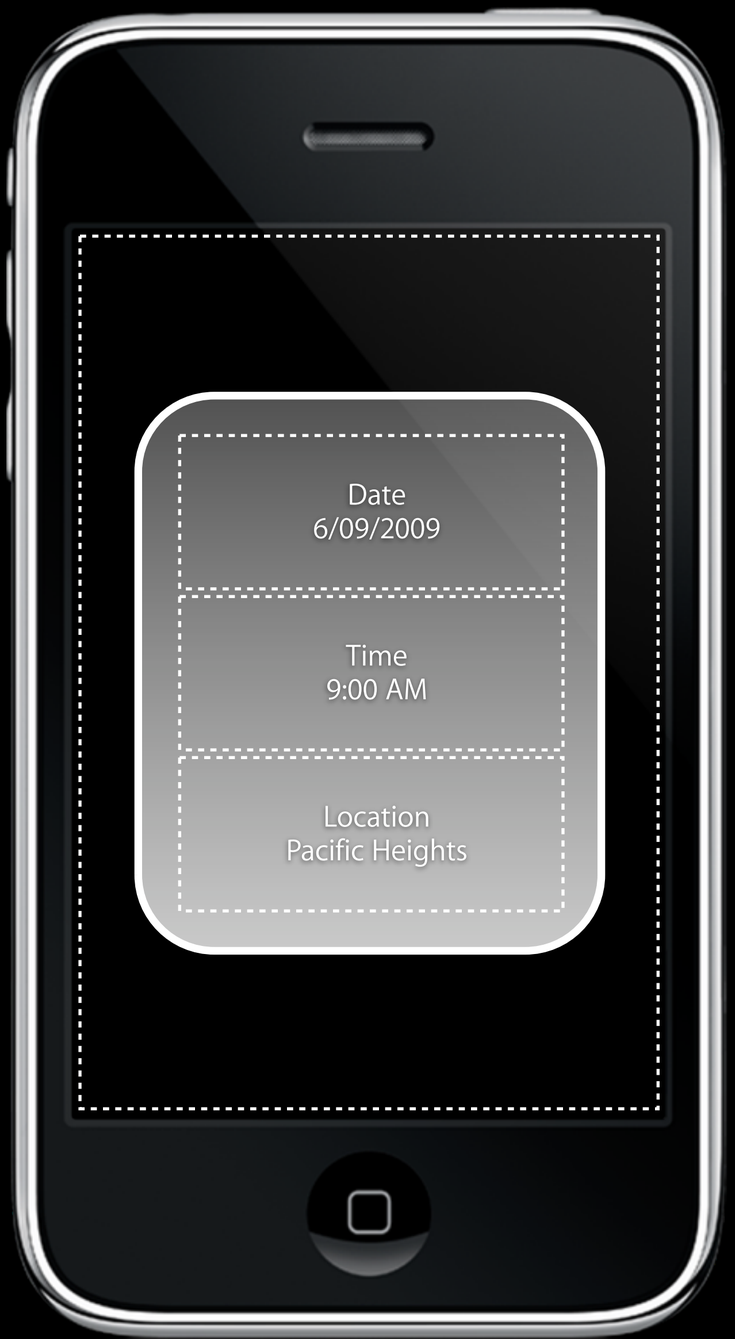
- Geometry
- Background
- Border



Layer Properties

CALayer

- Geometry
- Background
- Border
- Sublayers



Layer Properties

CALayer

- Geometry
- Background
- Border
- Sublayers
- Contents

Layer Properties

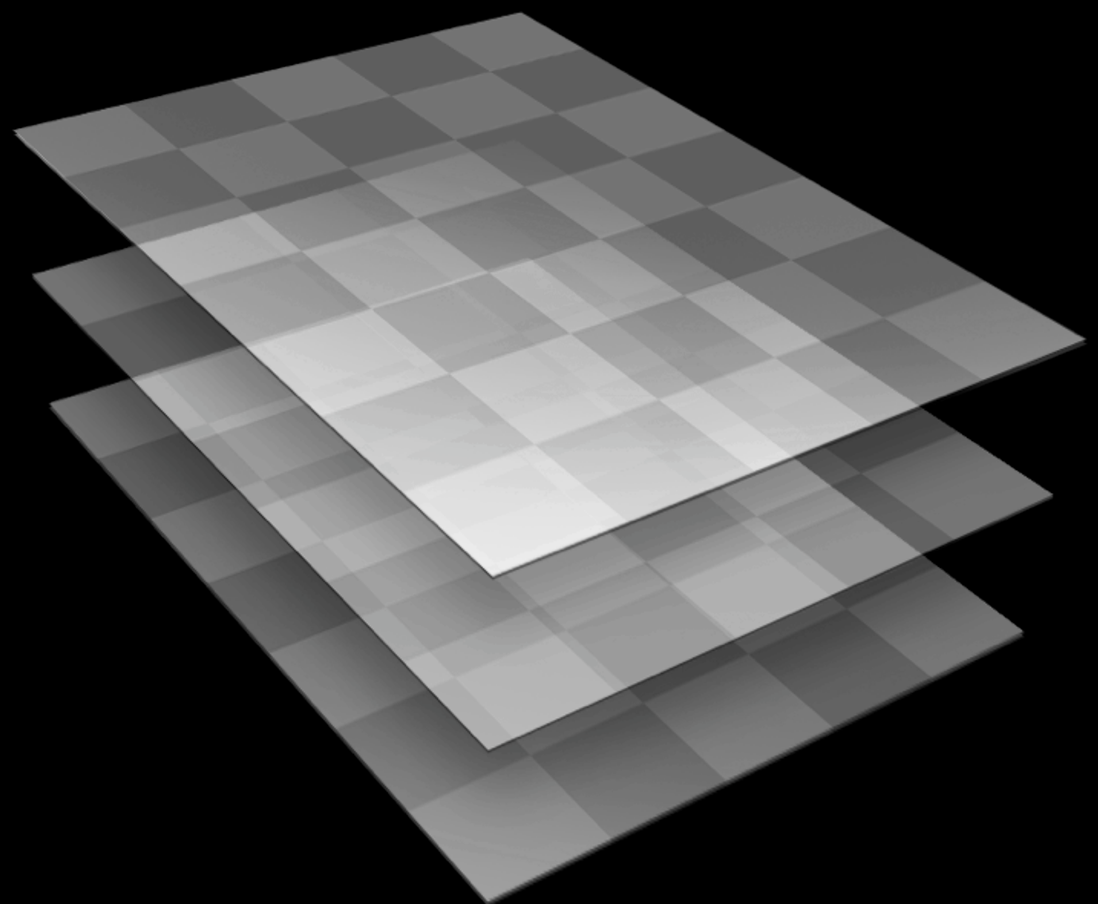
CALayer

- Geometry
- Background
- Border
- Sublayers
- Contents
- Opacity/Masking

Core Animation

Animate your user interface

- Animation can enhance the user experience
  - Adds realism and feedback for the user
  - Conveys a state change
  - Improves the visual production value of your interface
- Two overall concepts
  - Layers are composited
  - Animation is implicit



Animating Layers

"Implicit Animation" Model

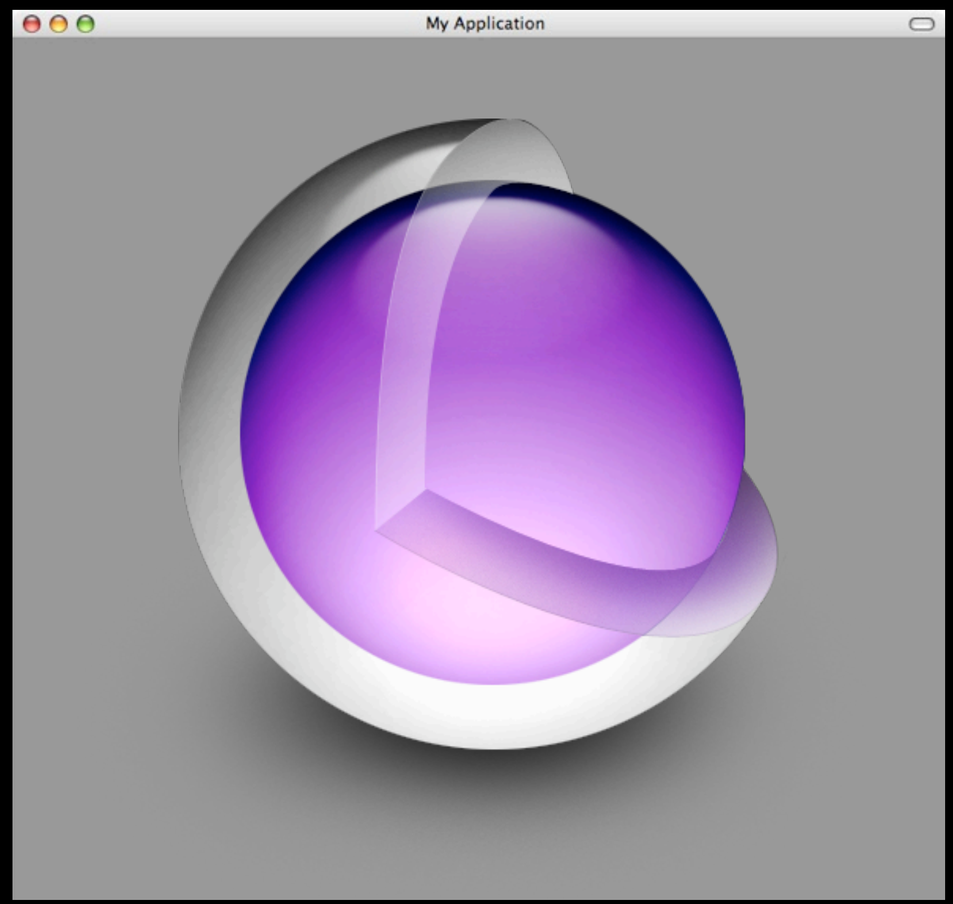
- Changing a layer property triggers an animation
- Animation begins asynchronously
- No waiting—control returns to your app immediately

Implicit Animations

MyApplication.m

```
logo.opacity = 0.0;
```

Code



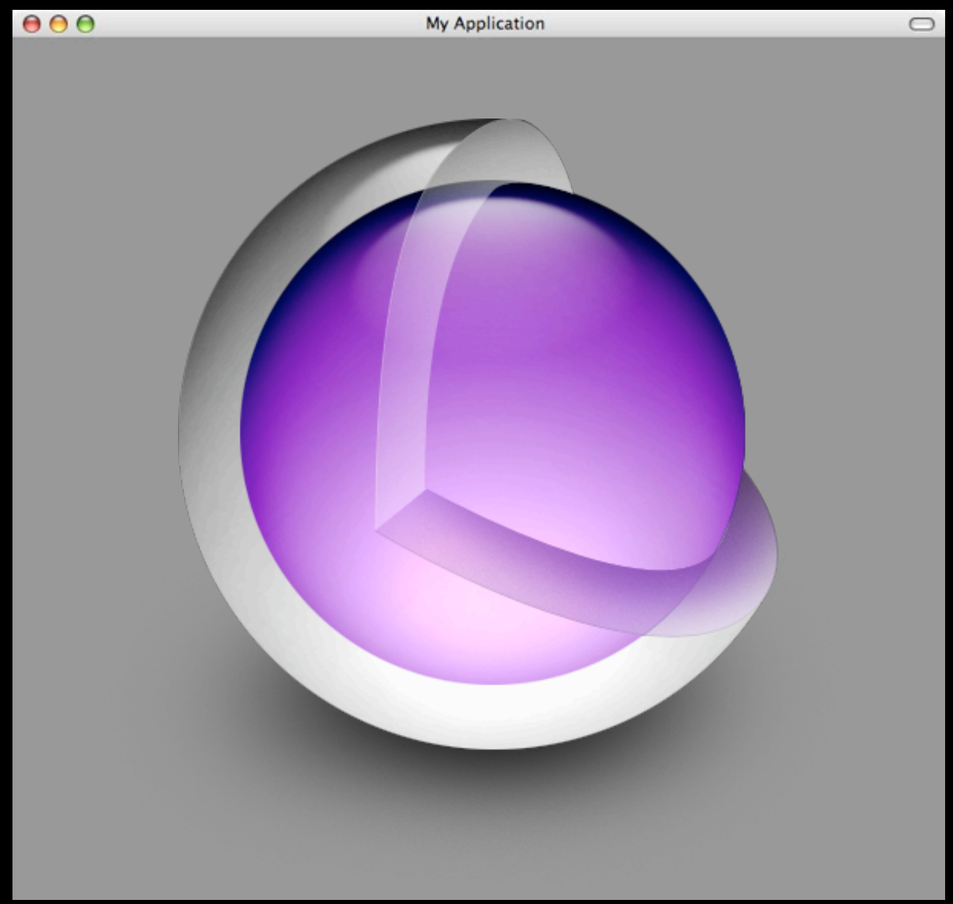
Result

Implicit Animations

MyApplication.m

```
logo.opacity = 1.0;
```

Code



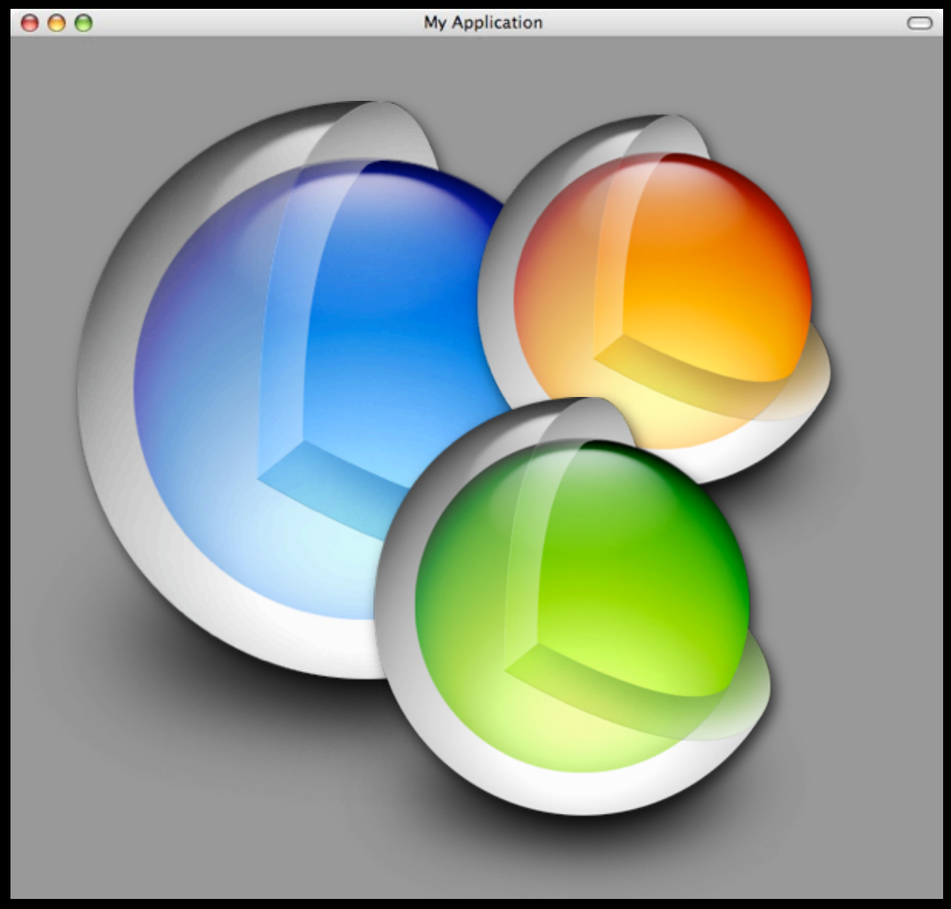
Result

Transactions

MyApplication.m

```
blueLayer.opacity = 0.0;
greenLayer.opacity = 0.0;
```

Code



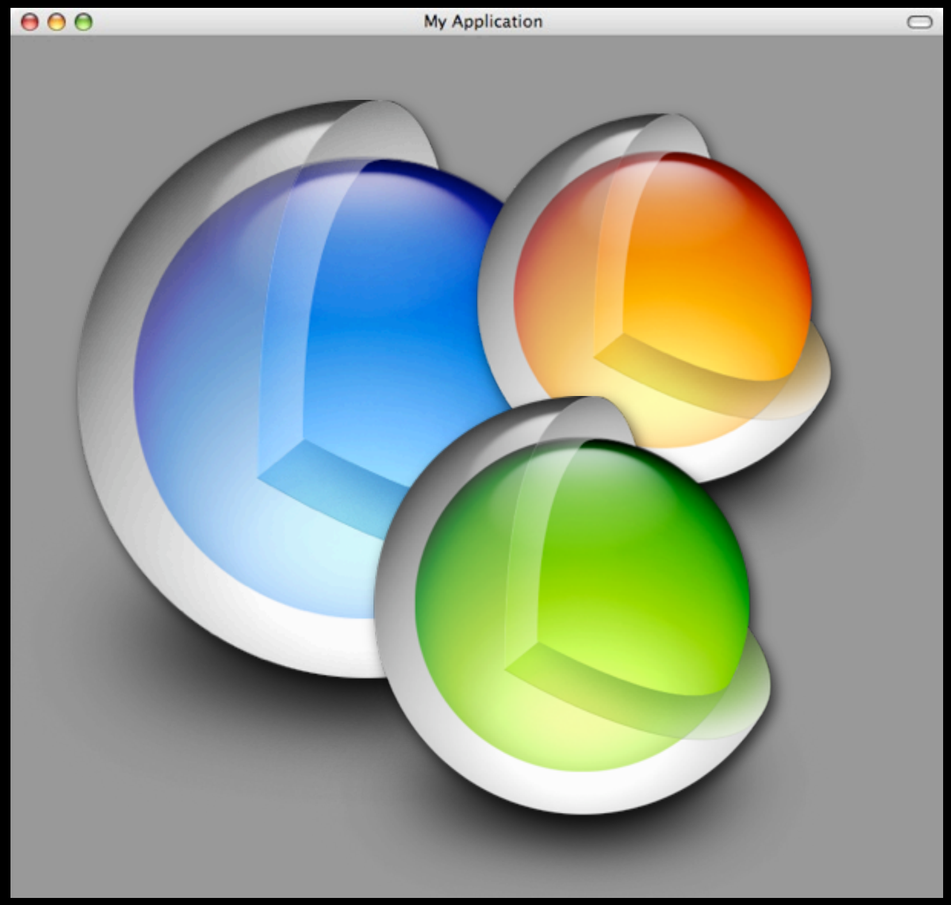
Result

Transactions

MyApplication.m

```
blueLayer.opacity = 0.0;
blueLayer.size = 0.0;
greenLayer.x = 500;
```

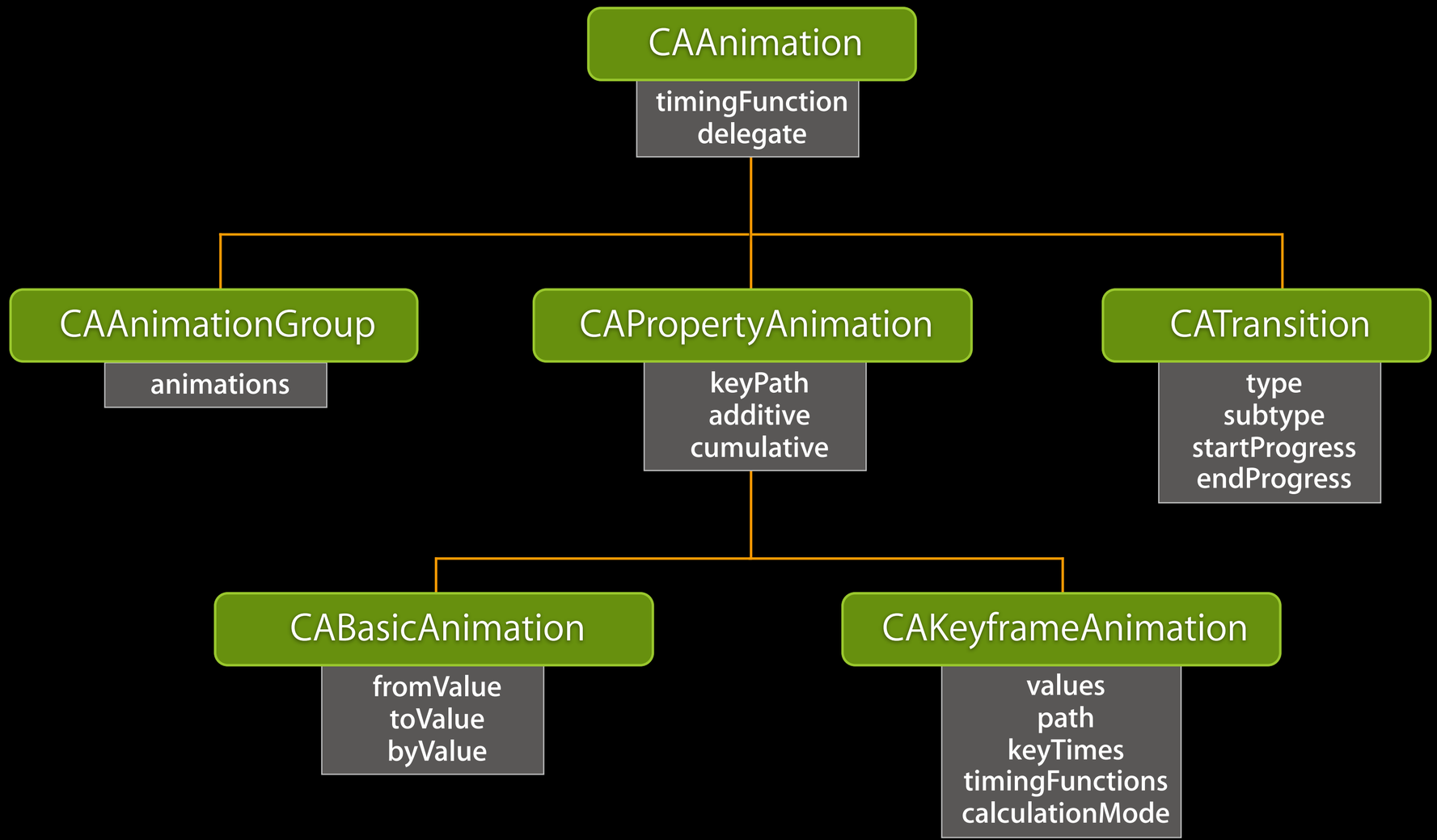
Code



Result

CAAnimations

Core Animation's expressive set of animation classes

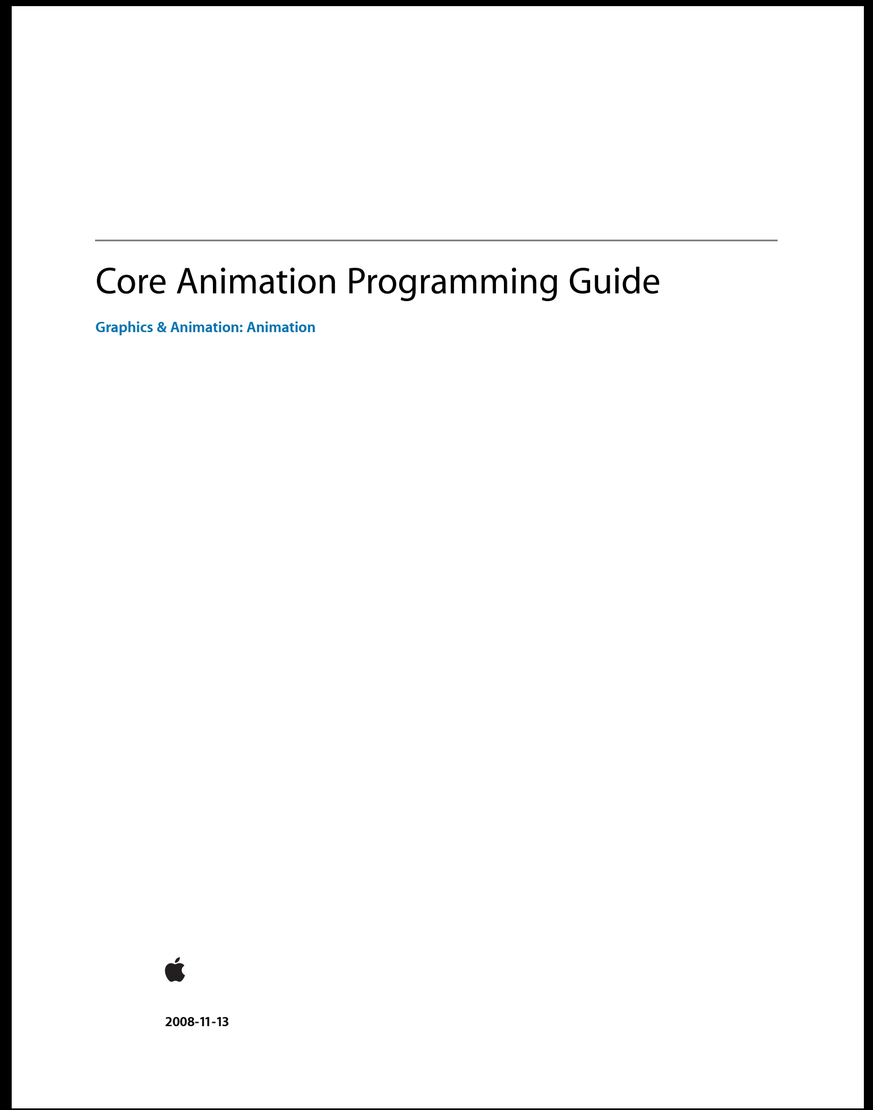


Demo

Core Animation

Learning More

- Core Animation Programming Guide
- Animation Types and Timing Programming Guide


Core Animation Programming Guide

Graphics & Animation: Animation

2008-11-13

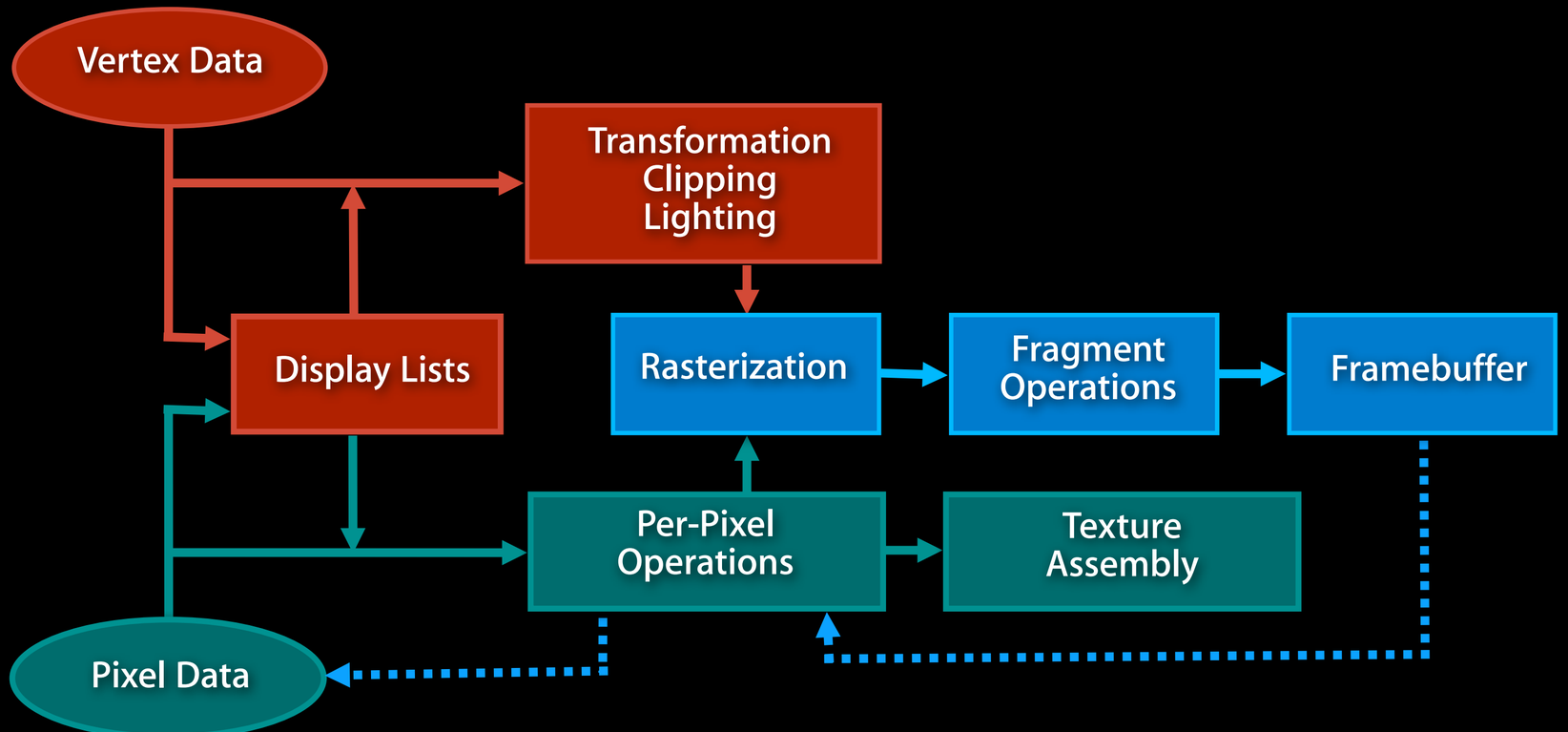
OpenGL ES

What Is OpenGL?

- A software interface to graphics hardware
  - Defined by the OpenGL specification
  - Not necessarily a 3D graphics API

What Does OpenGL Do?

- Takes vertices and pixels as input
- Applies visual operations to the data
- Produces pixel, or “fragment” data as output



Vertices, Geometry, and Fragments

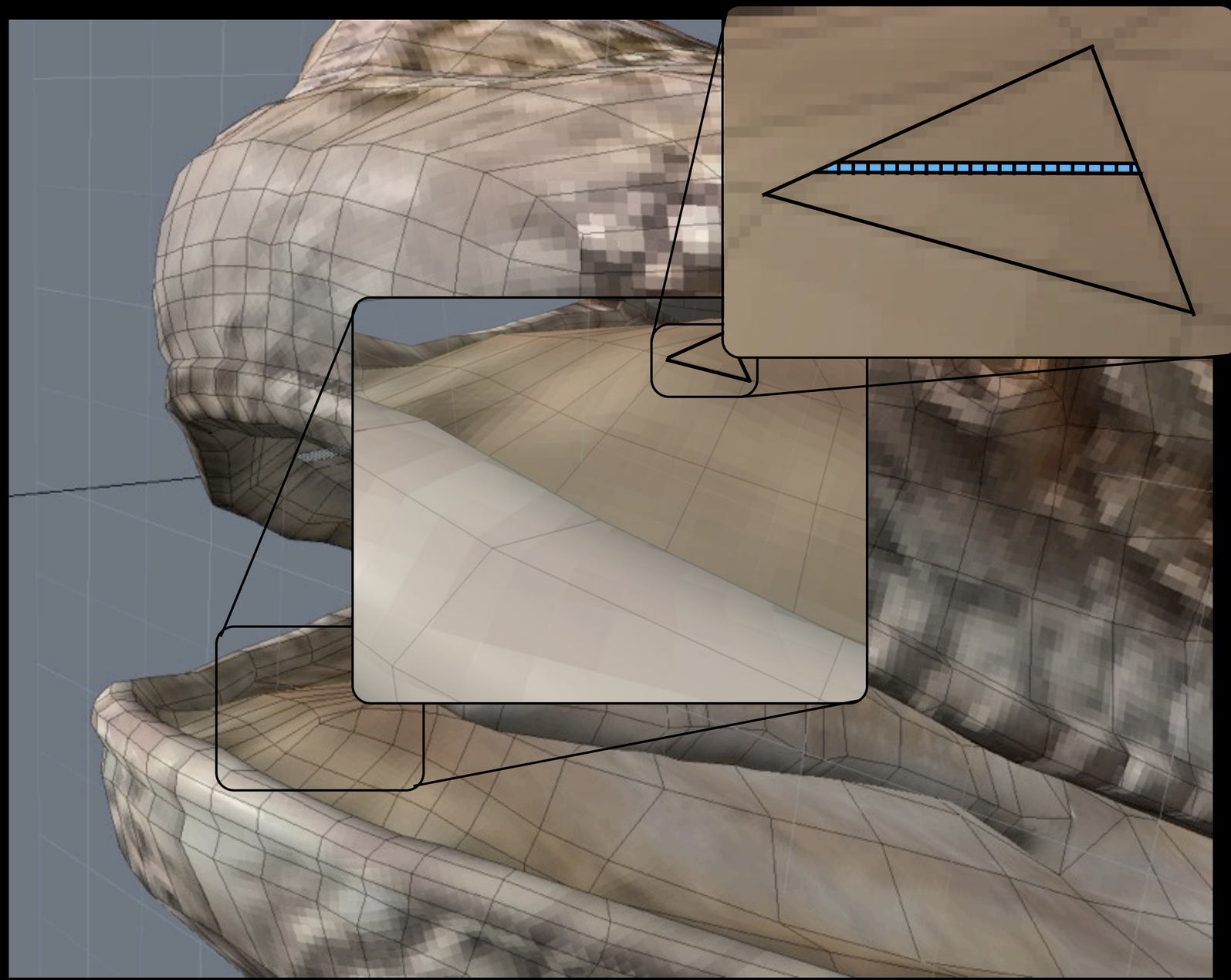


Image provided by Luxology Modo

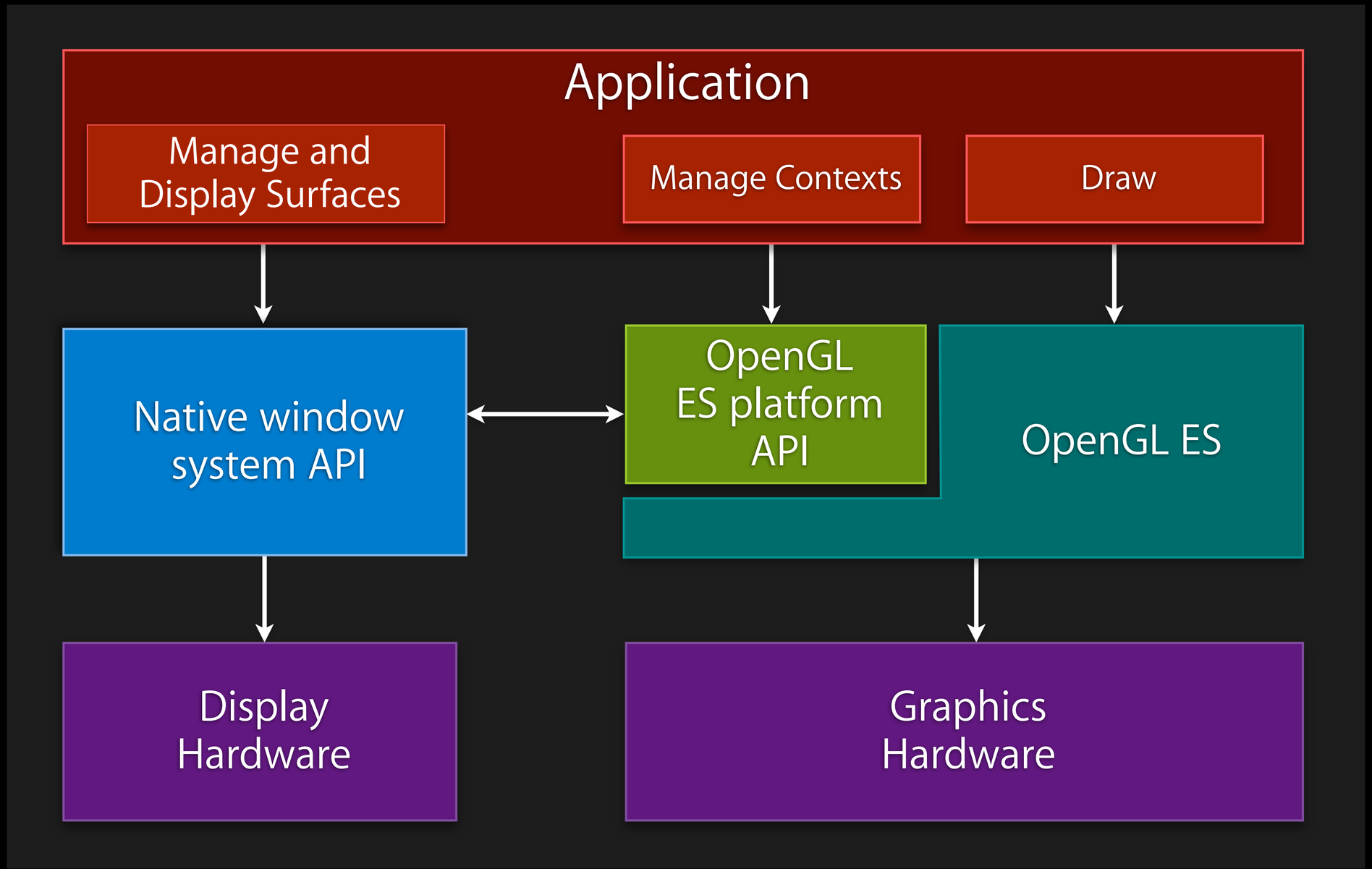
OpenGL ES

Overview

- A software interface to graphics hardware for embedded systems
  - High-performance 3D graphics API
  - The graphics API behind most games on iPhone
- Mobile version of the OpenGL standard
  - Defined by the OpenGL ES specification
  - Optimized subset of OpenGL
  - Targeted to embedded devices
  - Does not have a software fallback

OpenGL ES

Platform and Native Window System APIs



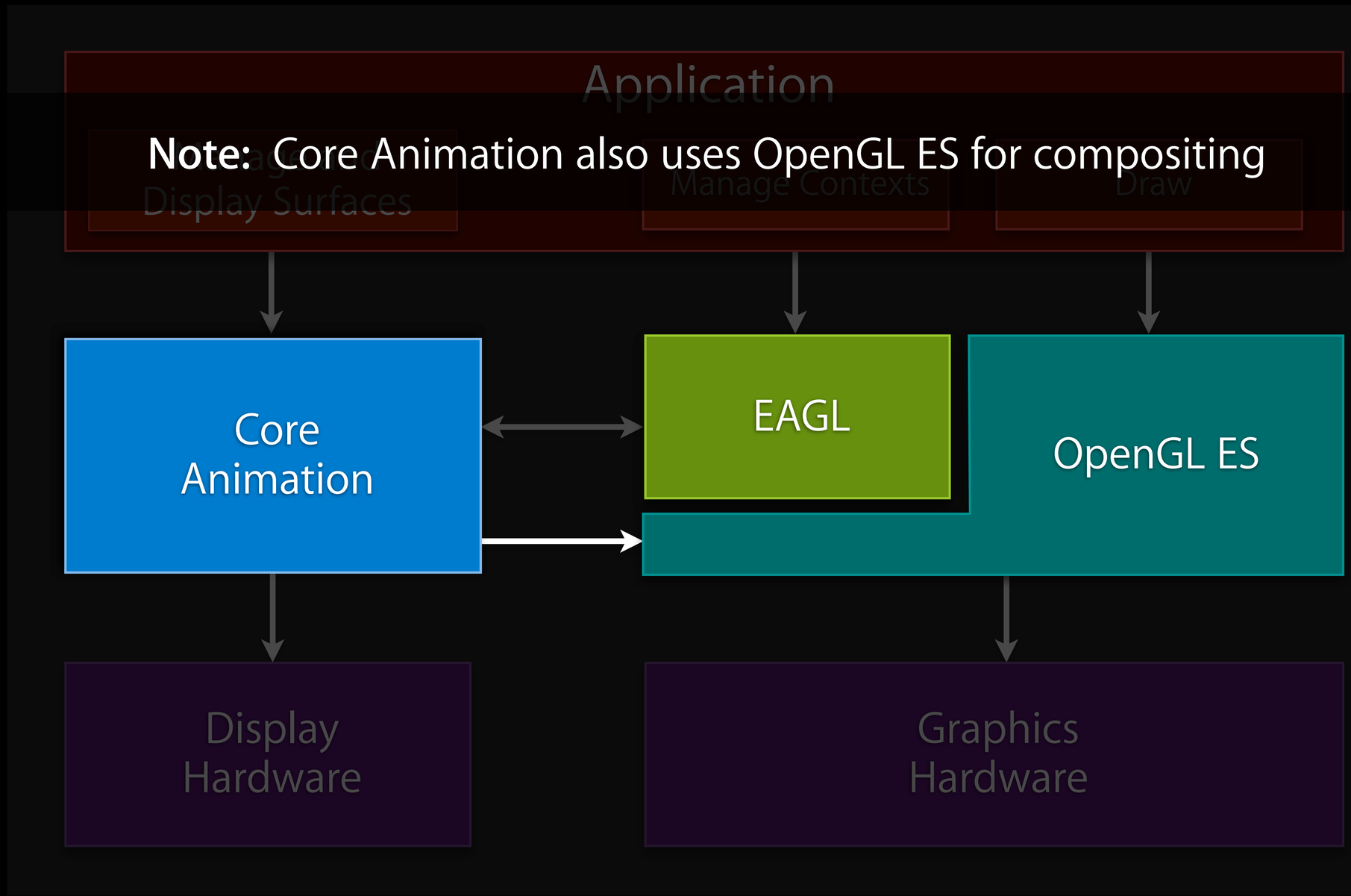
EAGL

iPhone OpenGL ES Platform API

- Defines OS interface for OpenGL ES
- Manages OpenGL ES contexts
- Works with Core Animation to display rendered content
- Provides drawable configuration

OpenGL ES on iPhone

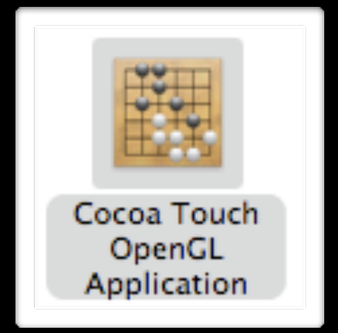
EAGL and Core Animation



Note: Core Animation also uses OpenGL ES for compositing

Learning the Basics

Getting started



- Cocoa Touch OpenGL ES Application Template
  - Starting point for an OpenGL ES-based application
  - EAGL setup, animation timer, render callback
  - Implement -drawView: to draw a frame

EAGLView.m

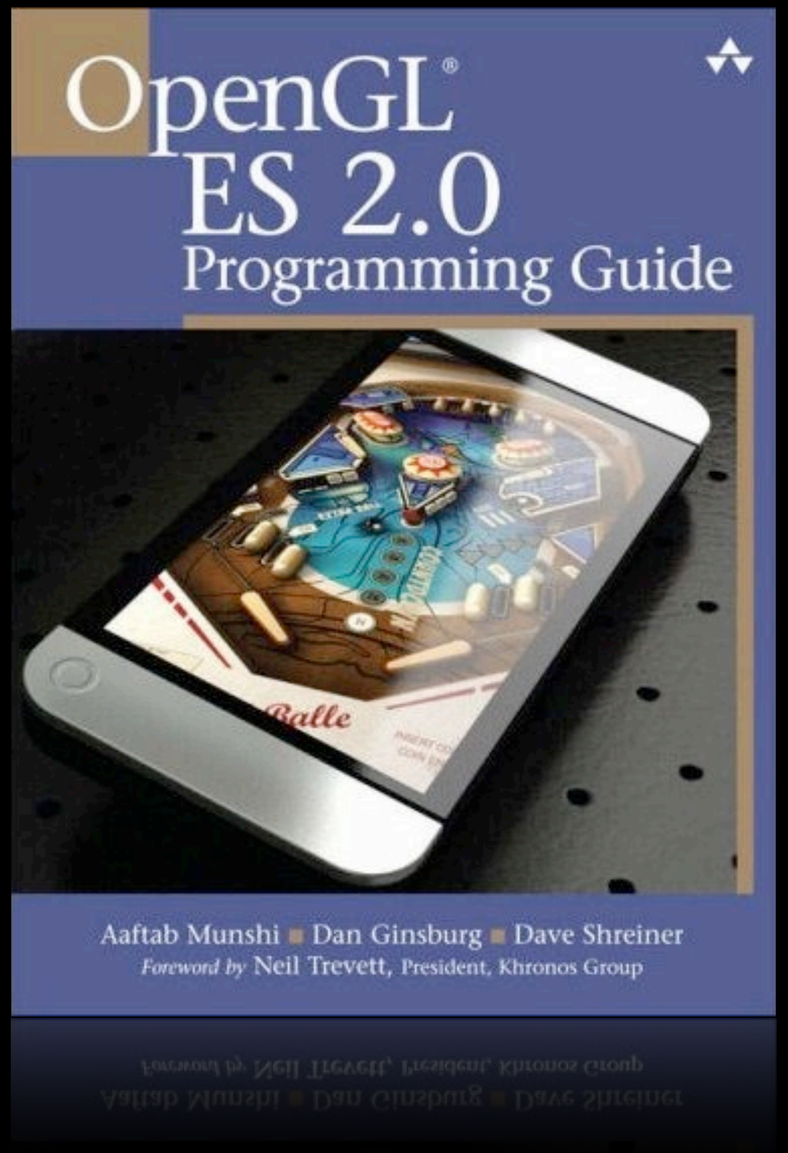
EAGLView.m:90 -drawView

```
83
84      [EAGLContext setCurrentContext:context];
85
86      glBindFramebufferOES(GL_FRAMEBUFFER_OES, viewFramebuffer);
87      glViewport(0, 0, backingWidth, backingHeight);
88
89      glMatrixMode(GL_PROJECTION);
90      glLoadIdentity();
91      glOrthof(-1.0f, 1.0f, -1.5f, 1.5f, -1.0f, 1.0f);
92      glMatrixMode(GL_MODELVIEW);
93      glRotatef(3.0f, 0.0f, 0.0f, 1.0f);
94
95      glClearColor(0.5f, 0.5f, 0.5f, 1.0f);
96      glClear(GL_COLOR_BUFFER_BIT);
97
98      glVertexPointer(2, GL_FLOAT, 0, squareVertices);
99      glEnableClientState(GL_VERTEX_ARRAY);
100     glColorPointer(4, GL_UNSIGNED_BYTE, 0, squareColors);
101     glEnableClientState(GL_COLOR_ARRAY);
```

Demo

Learning the Basics

- OpenGL ES 2.0 Programming Guide
  - The "Gold Book" (it's purple)
- http://developer.apple.com/iphone
  - OpenGL ES Programming Guide for iPhone
- OpenGL Programming Guide
  - The "Red Book"
- http://opengl.org
  - OpenGL ES specification, news, forums



Using the Right Tools

Using the Right Tools

UIKit and Quartz 2D

- Standard widget set
- Easy customization
- Tight integration

Using the Right Tools

Quartz 2D and Core Animation

- High-quality graphics
- Custom-rendered content
- Implicit and explicit animation

Using the Right Tools

UIKit and Core Animation

- Simple, easy-to-use interface elements
- Complex animation

Using the Right Tools

OpenGL ES and Core Animation

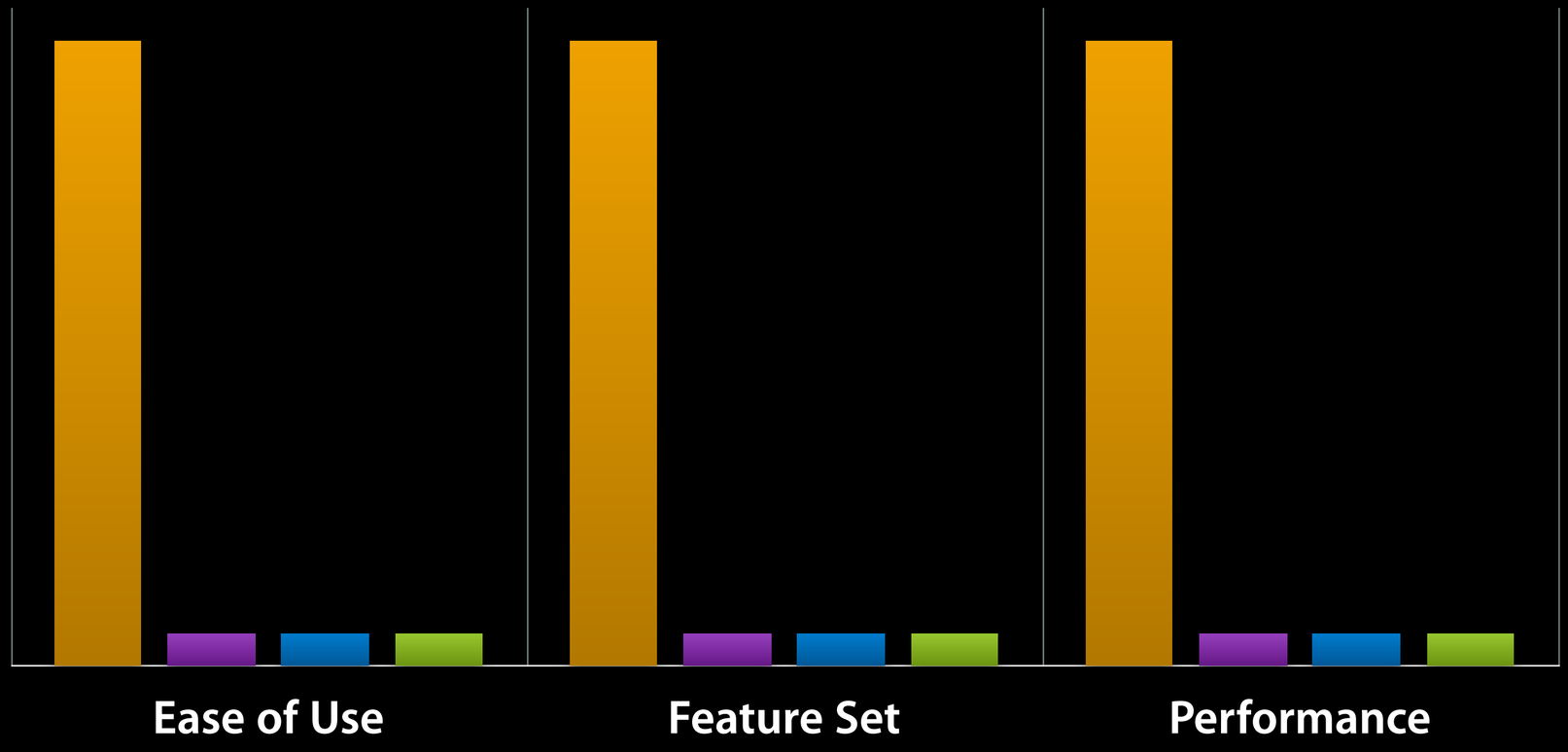
- Games, games, games
- 3D modeling
- 2D sprite animation

How Do I Choose?

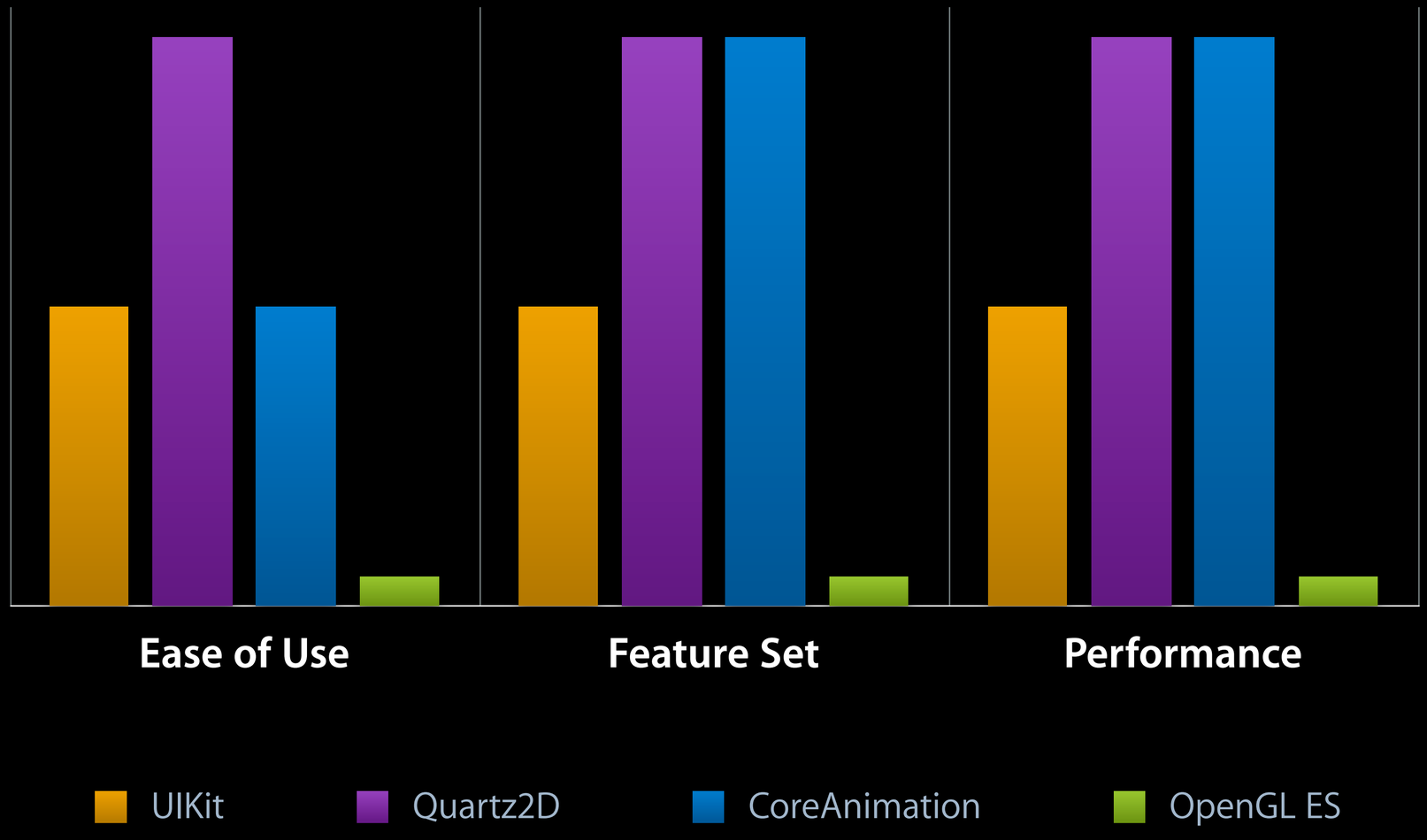
How Do I Choose?

- Different tradeoffs…
  - Performance, ease of use, feature set
- …for different needs
  - Standard and Custom GUI
  - 2D and 3D Graphics
  - Animation
  - Games

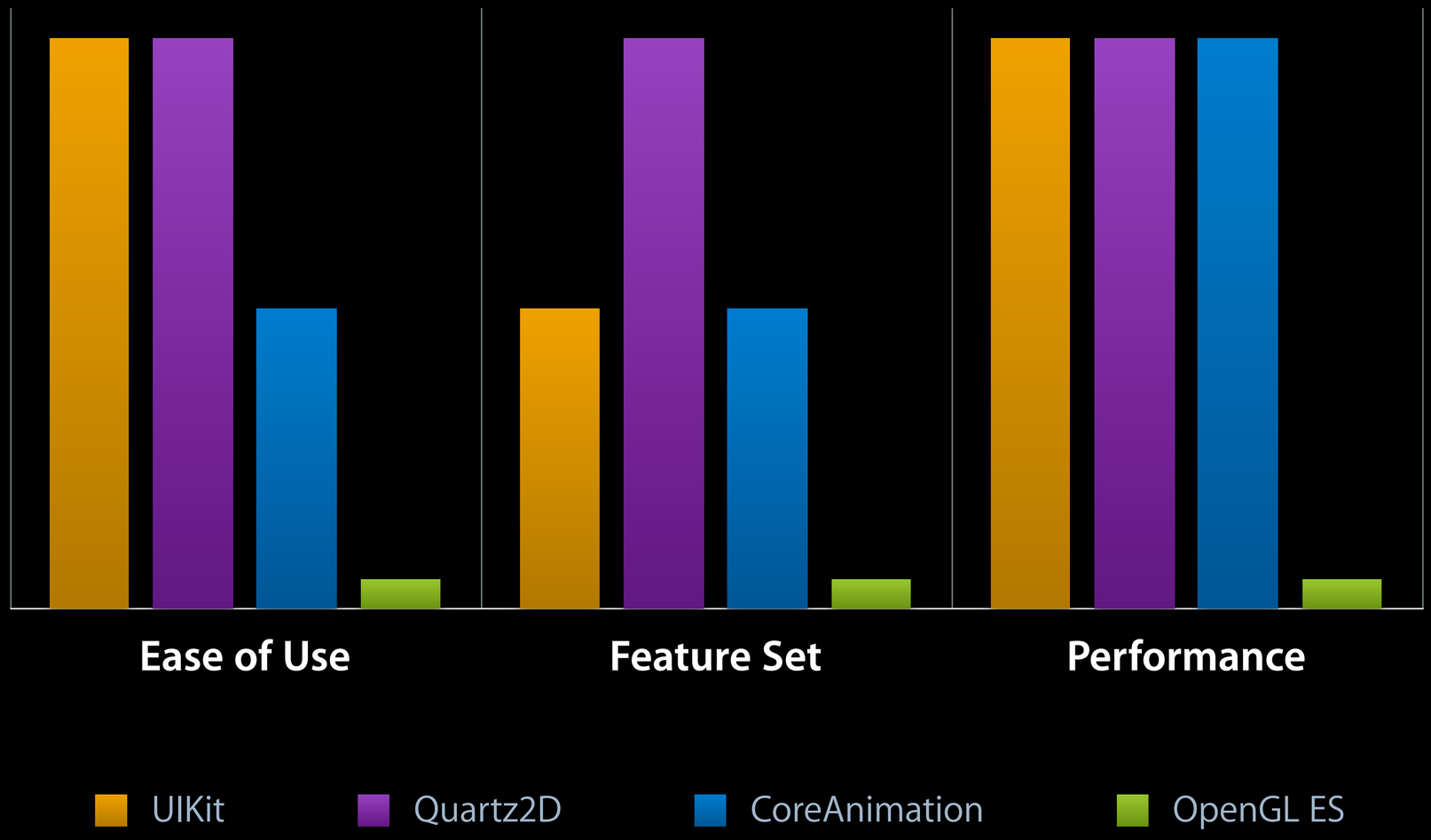
Standard GUI



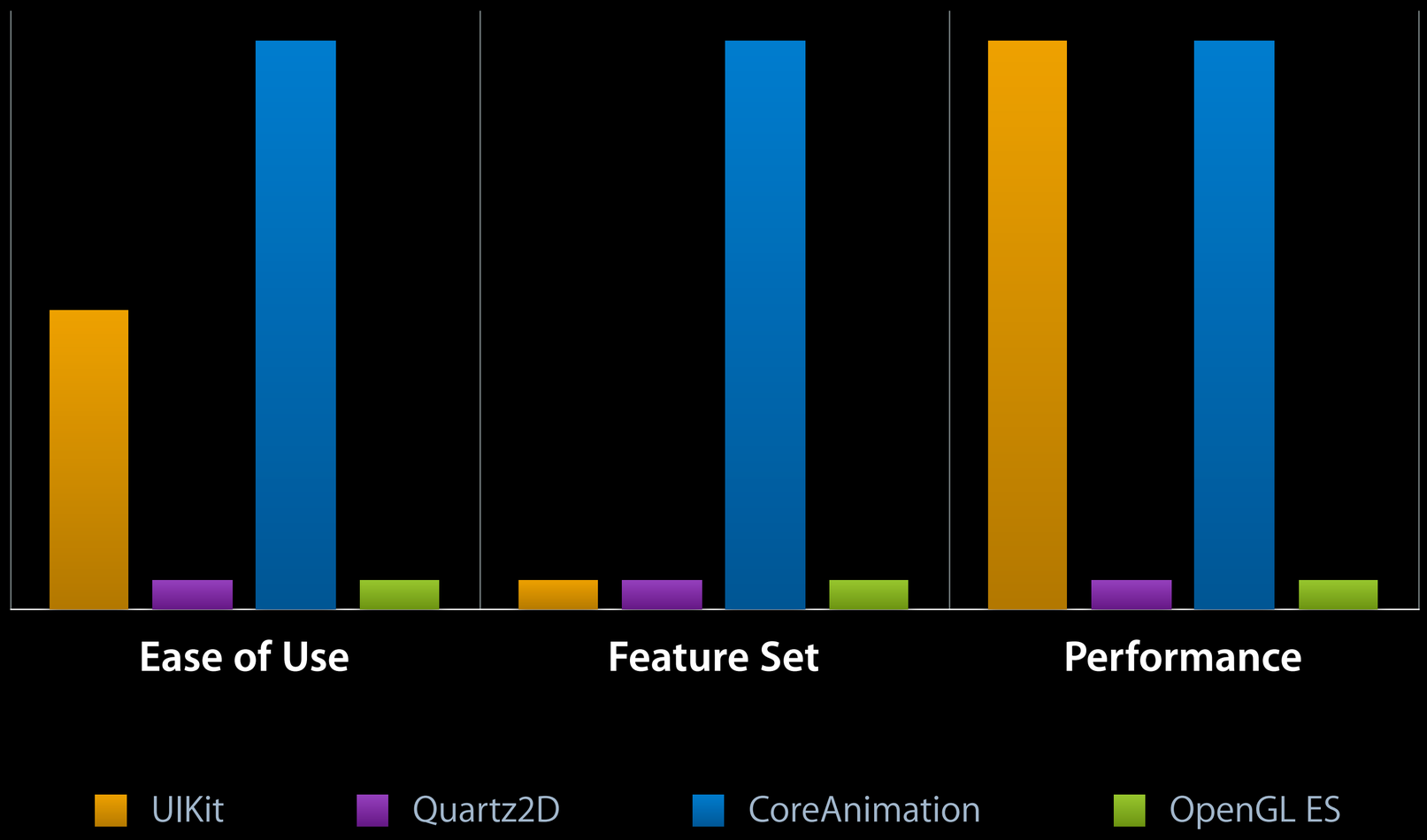
Custom GUI


Ease of Use
Feature Set
Performance
UIKit
Quartz2D
CoreAnimation
OpenGL ES

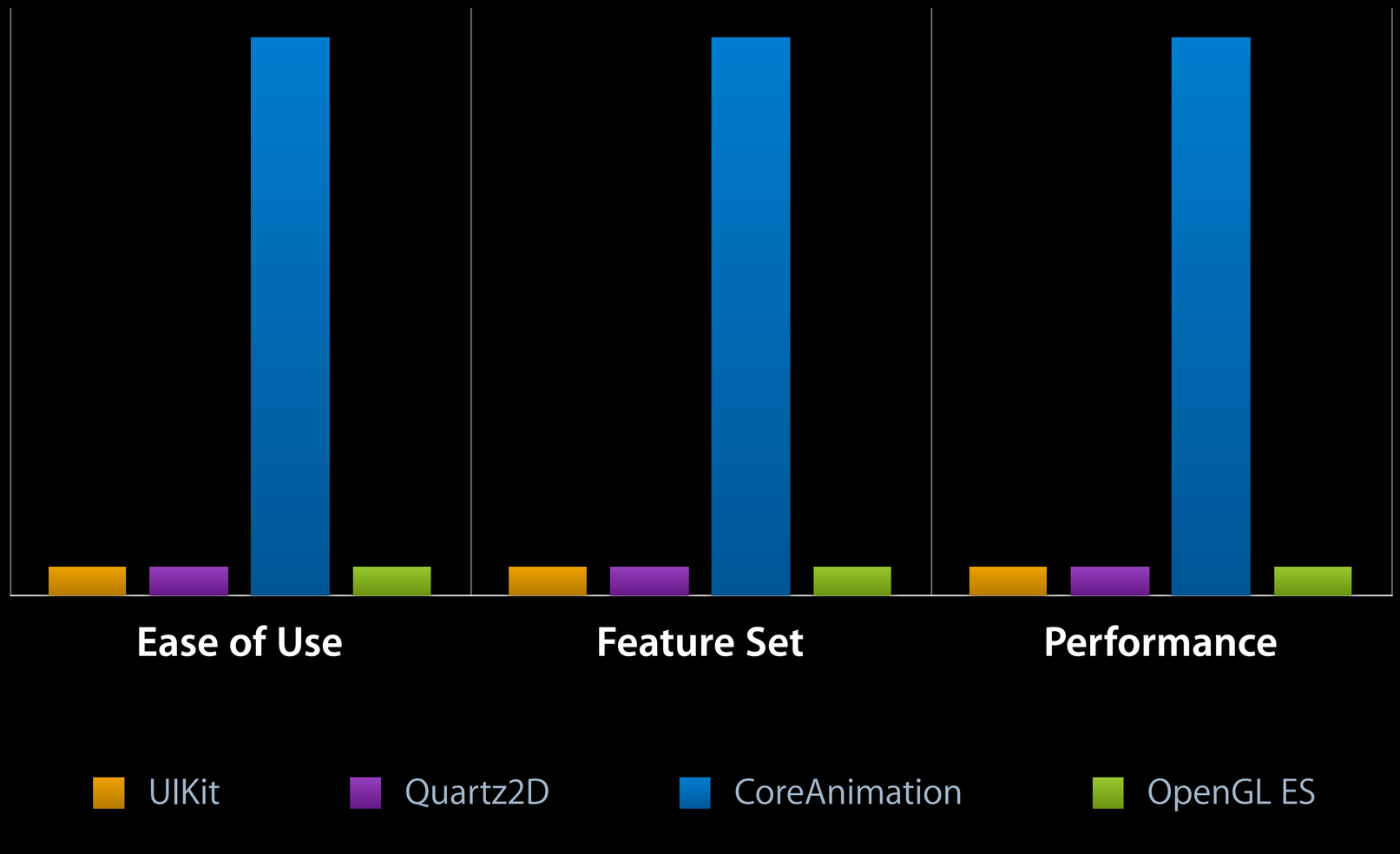
2D Graphics


Ease of Use
Feature Set
Performance
UIKit
Quartz2D
CoreAnimation
OpenGL ES

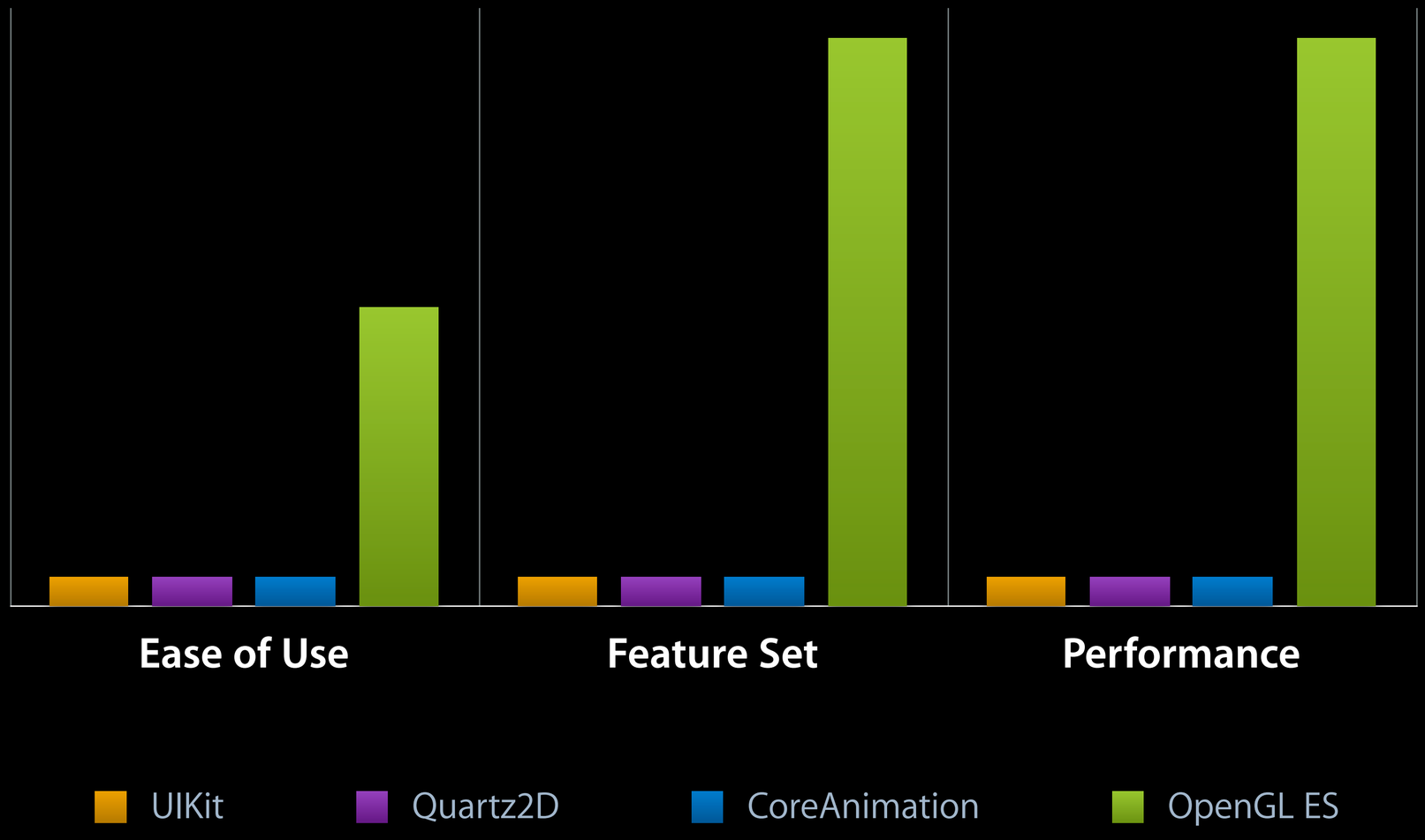
2D Animation



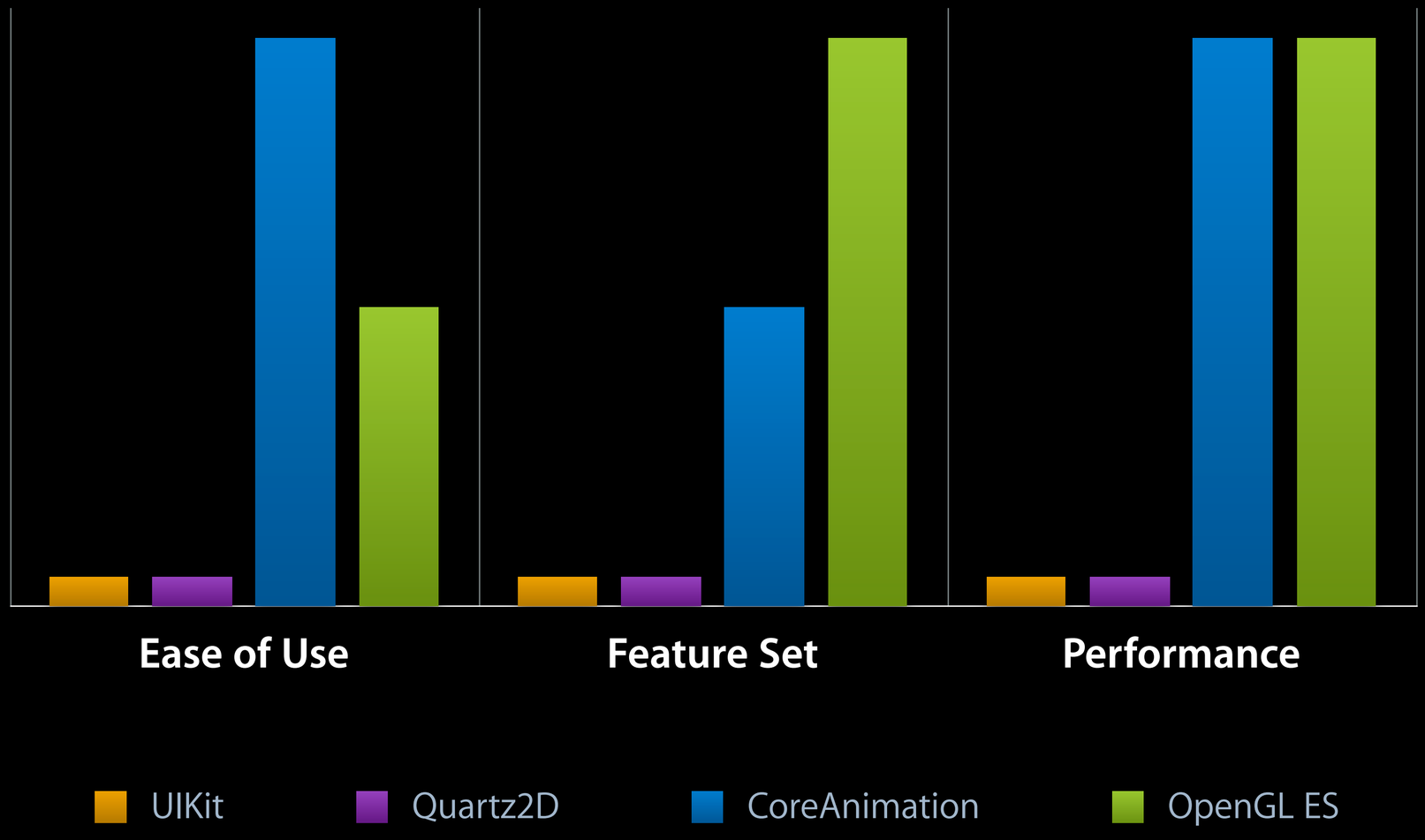
“2.5D” Graphics



3D Graphics


Ease of Use
Feature Set
Performance
UIKit
Quartz2D
CoreAnimation
OpenGL ES

3D Animation



And the Winner Is…

Standard GUI	UIKit
Custom GUI	Quartz2D
2D Graphics	Quartz2D
2D Animation	CoreAnimation
2.5D Graphics	CoreAnimation
3D Graphics	OpenGL ES
3D Animation	OpenGL ES

Really, Though, the Winner Is You!

More Information

Allan Schaffer
Graphics Technology Evangelist
aschaffer@apple.com

Documentation

iPhone Dev Center—Apple Developer Website
http://developer.apple.com/iphone

Related Sessions

Game Development for iPhone, Part 1	Presidio Tuesday 10:30AM
What's New in Cocoa Touch	Pacific Heights Tuesday 10:30AM
iPhone Views and Animations	Pacific Heights Tuesday 3:30PM
Core Animation Techniques for iPhone and Mac	Mission Wednesday 9:00AM
OpenGL ES Overview for iPhone OS	Presidio Wednesday 2:00PM

Labs

Graphics and Media Beginner's Lab	Graphics and Media Lab B Tuesday 12:00PM
Core Animation Lab	Graphics and Media Lab B Wednesday 12:00PM
Quartz 2D Lab	Graphics and Media Lab A Wednesday 12:00PM
OpenGL ES Lab	Graphics and Media Lab A Thursday 12:00PM
Game Development for iPhone Lab	iPhone Lab C Thursday 9:00AM

Q&A

WWDC Session Survey

You can rate your sessions on the WWDC Attendee Site.

Survey forms are located on each session description page at:

developer.apple.com/wwdc/attendee

WWDC09